IL MIO COMPUTER

CORSO PRATICO DI INFORMATICA: MICRO, MINI E PERSONAL COMPUTER



ISTITUTO GEOGRAFICO DE AGOSTINI

IL MIO
COMPUTER

IL MIO COMPUTER

CORSO PRATICO DI INFORMATICA: MICRO, MINI E PERSONAL COMPUTER

volume ottavo

ISTITUTO GEOGRAFICO DE AGOSTINI

Direzione: Achille Boroli - Adolfo Boroli
Direzione editoriale: Mario Nilo; *settore fascicoli:* Jason Vella

Redazione dell'edizione italiana a cura di:
Edizioni Remo Sandron - Firenze
Redazione dell'edizione inglese:
Richard Pawson (*Editor*); David Tebbutt (*Consultant Editor*); Chris Cooper (*Executive Editor*); David Whelan (*Art Director*); Gareth Jefferson, Roger Ford (*Collaborators*)
Designed and Produced by Bunch Partworks Limited
Traduzione a cura di: Daniel Quinn

Coordinamento grafico: Otello Geddo
Coordinamento fotografico a cura del Centro Iconografico dell'Istituto Geografico De Agostini

Titolo originale dell'opera: The home computer advanced course
© Orbis Publishing Limited, Londra, 1983.
© Istituto Geografico De Agostini S.p.A., Novara, 1986.
Stampato in Italia - IGDA Officine Grafiche, Novara, 1986.

Sommario

1681 La mente domina la materia
1684 Strutture dinamiche
1687 Il MicroPen
1689 Il DDI-1 della Amstrad
1692 Serrare i portelli di boccaporto
1695 Database
1696 I tendini del Sinclair
1698 Schermi multipli
1700 Impossible Mission
1701 Intelligenza e cervello
1704 Scegliere il giusto DBM
1706 La curvatura del braccio
1708 Database
1709 Il Wren Executive
1712 Ammutinamento a bordo
1714 Un duetto dinamico
1717 Il "beat" dei byte
1721 Sistemi di ricerca
1724 Approdo al Nuovo Mondo
1726 GOTO END
1729 Il Tandy 1000
1732 Database
1733 Un tracciatore digitale
1736 Disegni reticolari
1740 "Archon"
1741 Stampanti per Spectrum
1744 Controllo dinamico
1746 Ratto baratto
1748 Proiezioni strategiche
1752 Un prologo sul Prolog
1754 Montaggio del Tracer
1756 Rivoluzione grafica
1760 Database
1761 Metodi Scout e B-Star
1764 I comandi "transienti"
1766 Tracce per la memoria
1769 Database
1770 La Banana Interface
1772 Dichiarazioni di interesse
1774 Profitti e perdite
1776 I segreti dello Spectrum
1780 "Minder"
1781 Esperto a portata di chip
1784 I file nel CP/M
1786 Tracciamenti fantasiosi
1788 Database
1789 L'Atari 130XE
1792 Occhiate inquisitrici
1794 Nuovo Mondo I
1797 Canali di pensiero
1800 "Shadowfire"
1801 Come e cosa insegnare al computer?
1804 I dischetti del CP/M
1806 Effetti collaterali
1808 Il Ninbus RM
1812 Flussi di controllo
1815 Un'interfaccia MIDI
1818 Nuovo Mondo II
1820 Database
1821 Il sintetizzatore "Echo"
1823 Circuiti audio
1826 Un futuro promettente
1828 Codici genetici
1832 Analisi introspettiva
1834 Nuovo Mondo III
1837 Nastri e misure
1840 Football Manager
1841 Questioni di linguaggio
1843 Rappresentare la conoscenza
1846 Il mondo delle liste
1848 Linee di melodia
1852 Il Quick Data Drive
1854 Codice che genera codice
1856 Interrupt e utili routine
1861 Vedere e capire
1864 Liste e funzioni in LISP
1866 Regole di composizione
1869 L'Apricot F1e
1872 Ancora generatori di programmi
1874 I supercomputer
1876 Microdrive e porta seriale
1881 Riconoscere immagini
1884 Il Magic Mouse
1886 Liste attive
1889 Ottica e computer
1892 Hi-Fi con la MIDI
1895 Il DrLOGO
1897 Reti LAN e nuovi commandi BASIC
1901 Tra il dire e il fare...
1905 L'Omni-Read
1908 Il FORTH
1910 Ultime routine ROM

Indice per settore

Panorami



1681 La mente domina la materia
1701 Intelligenza e cervello
1721 Sistemi di ricerca
1748 Proiezioni strategiche
1761 Metodi Scout e B-Star
1781 Esperto a portata di chip
1801 Come e cosa insegnare ai computer
1828 Codici genetici
1841 Questioni di linguaggio
1843 Rappresentare la conoscenza
1861 Vedere e capire
1881 Riconoscere immagini
1901 Tra il dire e il fare...

Note d'Informatica



1684 Strutture dinamiche
1714 Un duetto dinamico
1726 GOTO END
1752 Un prologo sul Prolog
1792 Occhiate inquisitrici
1806 Effetti collaterali
1832 Analisi introspettiva
1846 Il mondo delle liste
1864 Liste e funzioni in LISP
1886 Liste attive
1908 Il FORTH

Software



1687 Il MicroPen
1700 Impossible Mission
1704 Scegliere il giusto DBM
1740 "Archon"
1744 Controllo dinamico
1764 I comandi "transienti"
1780 "Minder"
1784 I file nel CP/M
1800 "Shadowfire"
1804 I dischetti del CP/M
1826 Un futuro promettente
1840 Football Manager
1854 Codice che genera codice
1872 Ancora generatori di programmi
1895 Il DrLOGO

Hardware



1689 Il DDI-1 della Amstrad
1709 Il Wren Executive
1729 Il Tandy 1000

1741 Stampanti per Spectrum
1770 La Banana Interface
1789 L'Atari 130XE
1808 Il Ninbus RM
1821 Il sintetizzatore "Echo"
1852 Il Quick Data Drive
1869 L'Apricot F1e
1874 I supercomputer
1884 Il Magic Mouse
1889 Ottica e computer
1905 L'Omni-Read

Programmazione BASIC

1692 Serrare i portelli di boccaporto
1712 Ammutinamento a bordo
1724 Approdo al Nuovo Mondo
1746 Ratto baratto
1774 Profitti e perdite
1794 Nuovo Mondo I
1818 Nuovo Mondo II
1834 Nuovo Mondo III

Linguaggio macchina

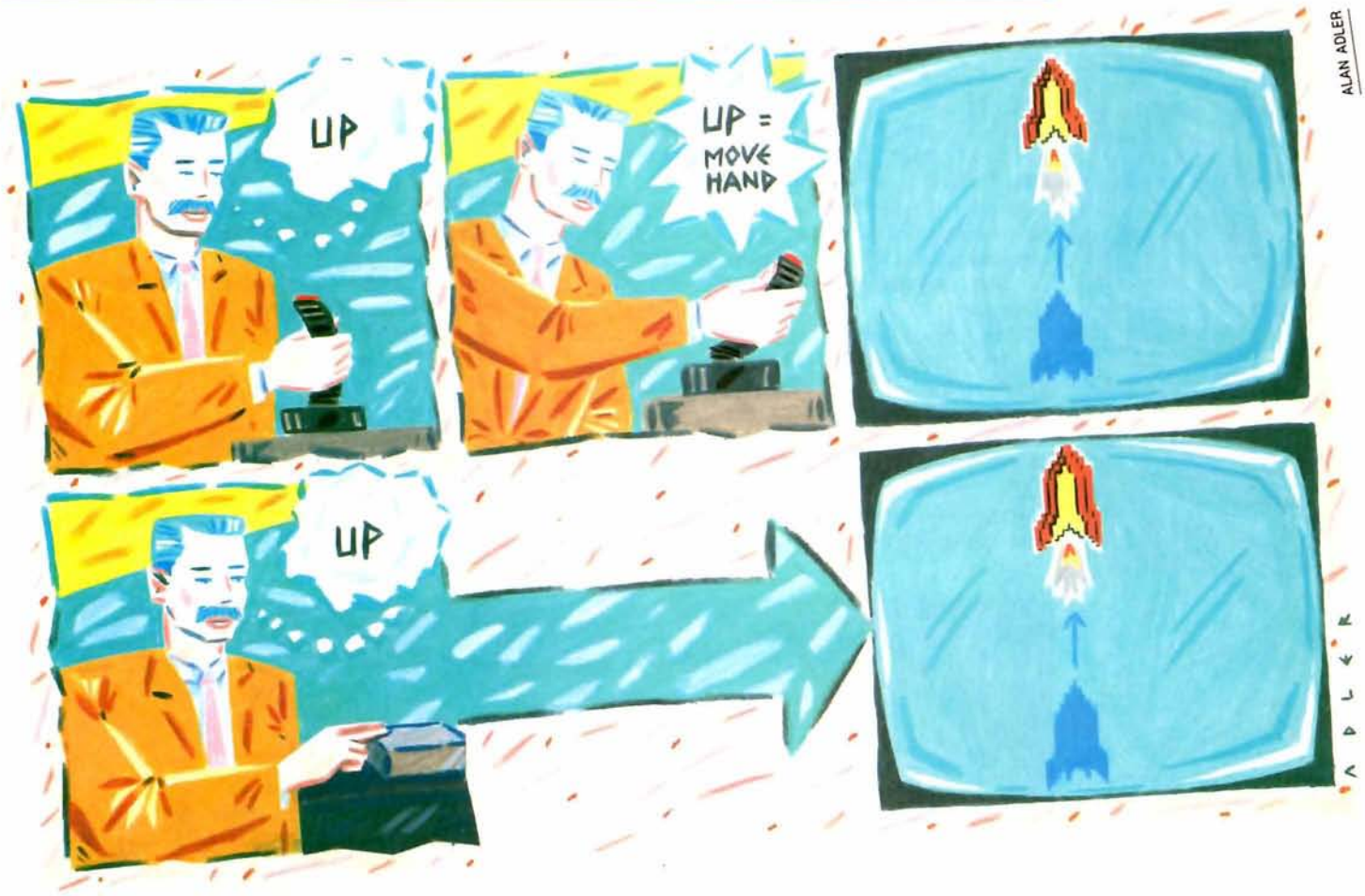
1695 Database
1698 Schermi multipli
1708 Database
1717 Il "beat" dei byte
1732 Database
1736 Disegni reticolari
1756 Rivoluzione grafica
1760 Database
1769 Database
1772 Dichiarazioni d'interesse
1776 I segreti dello Spectrum
1788 Database
1797 Canali di pensiero
1812 Flussi di controllo
1820 Database
1837 Nastri e misure
1856 Interrupt e utili routine
1876 Microdrive e porta seriale
1897 Reti LAN e nuovi commandi BASIC
1910 Ultime routine ROM

Laboratorio

1696 I tendini del Sinclair
1706 La curvatura del braccio
1733 Un tracciatore digitale
1754 Montaggio del Tracer
1766 Tracce per la memoria
1786 Tracciamenti fantasiosi
1815 Un'interfaccia MIDI
1823 Circuiti audio
1848 Linee di melodia
1866 Regole di composizione
1892 Hi-Fi con la MIDI

La mente domina la materia

Controllare un computer tramite il solo pensiero non è più un fatto di fantascienza



Fin dagli inizi, il principale dispositivo usato per immettere dati nei computer è stato la tastiera. Ancora oggi, almeno finché non si perfezioneranno sistemi capaci di riconoscere la voce umana, la tastiera rimane il mezzo più adatto per la digitazione di testi e, in modo più limitato, di numeri. Tuttavia, questo mezzo non può essere considerato il migliore per quanto riguarda altri tipi di informazioni: è il caso delle direttive di movimento in programmi di simulazione in tempo reale o in molti giochi. A questi fini sono molto più adatti joystick, mouse, track-ball, penne luminose o schermi sensibili al tatto. La lettura di grandi quantità di dati numerici, invece, è facilitata dall'uso di un lettore di codici a barre.

Tutti questi dispositivi di immissione, però, presentano un lato negativo comune: sono meccanismi "indiretti". Anche i più sofisticati joystick, coi quali il controllo di un oggetto sullo schermo (un'astronave in un gioco, ad esempio) è pressoché immediato, si pongono quale intermediario tra quanto l'operatore desidera e quanto accade effettivamente sullo schermo del computer. In altre parole, il pensiero "movimento in su" va tradotto mentalmente in "movimento in avanti del joystick" e, infine, nell'effettivo azionamento del joystick.

Per ottenere un'azione più immediata e diretta andrebbe dunque eliminato questo passaggio intermedio: arrivare cioè a "pensare" un movimento e fare in modo che il computer interpreti direttamente tale pensiero.

Certo sarebbe comodo poter scrivere una lettera semplicemente pensando le parole, senza dover usare le mani, ma il computer

Giochi mentali

Il principio del "Mindlink" elimina la fase meccanica intermedia tra il pensiero e il controllo del computer: piccole variazioni di elettricità, registrate sulla pelle del giocatore tramite appositi elettrodi, vengono misurate e interpretate dal software del computer. Questo sistema offre una maggior velocità di risposta rispetto ai sistemi tradizionali

NOTE MENTALI



Nella foto si vede un prototipo di sistema GSR collegato ad un Apple, ma sono state eseguite prove anche su IBM e Commodore. Il problema è di perfezionare il software di controllo che funge da interprete dei segnali letti dall'interfaccia. Roger Dilts, presidente della Behavioral Engineering e creatore del primo software GSR, è particolarmente interessato alla possibilità di combinare il Mindlink con la NLP (Neuro-Linguistic Programming), ossia la programmazione neuro-linguistica, una branca della psicologia che si occupa dello studio dell'apprendimento. Le condizioni emotive del soggetto applicato allo studio possono essere infatti campionate dal software GSR, individuando quando si presentano situazioni di affaticamento mentale e adeguando così il flusso del programma di apprendimento. Roger Dilts è convinto che il software didattico sarà presto in grado di adeguarsi al ritmo di apprendimento di ciascun allievo, misurando le reazioni emotive di fronte alle varie fasi dello studio

telepatico è ancora ben lontano dalla realtà di tutti i giorni. Tuttavia, oggi è già possibile cimentarsi in qualche gioco (del tipo di *Defender*, ad esempio) controllando i movimenti tramite il pensiero. Ciò è reso possibile ricorrendo al concetto chiamato "Mindlink", ossia alla "connessione mentale".

Secondo questo concetto il pensiero (o più precisamente le variazioni fisiologiche che risultano dai cambiamenti delle configurazioni mentali) è usato per controllare un dispositivo elettronico. Questo viene interfacciato ad un computer né più né meno come un normale joystick; il suo funzionamento è basato su quello che viene chiamato il "fenomeno GSR", in cui i mutamenti emozionali di una persona si manifestano tramite impercettibili (ma misurabili) variazioni di conduttività elettrica nella pelle. In sostanza si tratta dello stesso principio adottato nelle "macchine della verità", solo che in questo caso l'individuo è collegato, tramite elettrodi applicati sul corpo, ad un misuratore di resistenza, la cui lettura viene poi passata al computer. Un apposito programma interpreta i segnali, convertendoli in movimenti sullo schermo. La Behavioral Engineering, una ditta californiana, si è specializzata in questo tipo di applicazioni anche nel settore ricreativo, producendo appunto una versione del gioco *Defender* controllato dal pensiero. Anche altre grosse ditte, fra le quali l'Atari, lavorano da tempo a simili progetti, ma solo recentemente si sono avuti risultati di qualche rilievo pratico.

Nel gioco *Defender*, versione originale, si tratta di controllare un'astronave in orbita attorno ad un pianeta: lo scopo è quello di abbattere navicelle nemiche senza essere colpiti, avendo cura di non precipitare sul pianeta. Nella versione della Behavioral Engineering il controllo è limitato all'altitudine della nave spaziale, tuttavia si tratta di un controllo mentale, realizzato per mezzo di un'interfaccia GSR applicata ad un Apple IIe. Basta che il giocatore inserisca il dito indice e medio in un apparecchietto simile a un "mouse", per trasmettere al computer le variazioni di conduttività superficiale della pelle. Una resistenza elettrica bassa provoca un innalzamento dell'astronave e viceversa. Ciò consente di allineare il veicolo spaziale ai bersagli in avvicinamento.

Il termine "Mindlink" che pare sia stato coniato dalla Atari, può dar luogo a malintesi in quanto la connessione, in effetti, è con il sistema nervoso e non con la mente. Tutta-

GSR: IL PRINCIPIO DEL MINDLINK

Il principio operativo del Mindlink è noto con tre diverse sigle: GSR (Galvanic Skin Response, ossia risposta galvanica della pelle), PGR (PsycoGalvanic Reflex, ossia riflesso psico-galvanico) e EDR (ElectroDermal Reflex, ossia riflesso elettrodermico).

Il fenomeno GSR si riferisce alle variazioni di conduttività elettrica della pelle, che corrispondono a mutamenti nello stato emotivo del soggetto sotto esame.

Le esperienze hanno dimostrato che più la persona è tesa, minore è la resistenza elettrica nella pelle. Una delle applicazioni più note del fenomeno GSR è la cosiddetta "macchina della verità", o "lie-detector".

Benché lo studio del fenomeno GSR sugli animali e sugli uomini risalga al XIX secolo, non si sono ancora scoperte le vere cause che lo provocano. Una delle prime ipotesi formulate afferma che è il sudore prodotto in stati di particolare ansietà o eccitazione ad agire semplicemente da conduttore elettrolitico, facendo diminuire la resistenza elettrica superficiale della pelle. Tuttavia, recenti studi sembrano contraddire questa spiegazione un poco semplicistica.

Resta il fatto che, qualunque ne sia la causa, il fenomeno GSR è strettamente correlato al grado di tensione presente nel sistema nervoso simpatico, che a sua volta dipende dal sistema nervoso centrale e, in definitiva, dal cervello. Ecco quindi la possibilità di controllare un computer "usando il pensiero": il cervello formula un pensiero e il sistema nervoso provoca un cambiamento che si manifesta con variazioni nella conduttività superficiale della pelle. Poiché qualsiasi variazione elettrica è utilizzabile come immissione per un'interfaccia, il resto è semplice



IAN DOBBIE

via, almeno in questa particolare applicazione, la distinzione è alquanto indefinita. Molte persone raggiungono un buon livello di controllo dopo una ventina di minuti di esercizio, ad altre basta semplicemente "pensare" la direzione del movimento, per ottenere immediatamente i risultati desiderati. In entrambi i casi, raramente il soggetto si preoccupa del modo in cui avviene effettivamente il fenomeno.
Oltre ai giochi, naturalmente, il "Mindlink" è utilizzabile in altre applicazioni. Persone affette da paralisi, ma in grado di generare coscientemente segnali GSR, possono usufruire di sistemi simili per controllare i più svariati meccanismi. È già stato realizzato, a tale scopo, un sistema che permette il controllo di un robot Topo tramite un computer Apple. Un'altra interessante applicazione è nel settore spaziale, in condizioni di gravità zero, dove esercitare pressioni su controlli meccanici è relativamente difficile: il sistema GSR si dimostra molto più pratico.

Il software reagisce agli umori

Tra le applicazioni forse più fantasiose, c'è quella di usare il fenomeno GSR per campionare lo stato emotivo di un soggetto durante l'utilizzazione di un computer, particolarmente durante cicli di apprendimento. Appositi sensori possono individuare quando la persona passa dalla normale condizione di rilassamento ad una di tensione o di affaticamento: il software, a questo punto, può comportarsi di conseguenza e suggerire, magari, una breve interruzione del lavoro. Negli uffici del futuro gli impiegati saranno dunque legati "col guinzaglio" al computer? C'è già chi vede in questo innovamento un temibile segnale della "computer-dipendenza" che sembra dover affliggere la società del domani.
La diffusione delle applicazioni del fenomeno GSR dipende molto dalla realizzazione di apparecchiature in grado di interpretare con maggiore precisione i segnali rilevati e di offrire una risposta più immediata. Infatti, il tempo che intercorre tra un evento e la generazione di un segnale GSR è di circa due secondi e tale impulso perdura dai due ai dieci secondi. Le apparecchiature attuali aggirano l'ostacolo misurando il tasso di variazione del segnale, piuttosto che la sua intensità, quindi resta ancora molto cammino da percorrere. Anche l'interpretazione dei segnali è per il momento empirica: per adesso si può stabilire che una rapida diminuzione di resistenza elettrica nella pelle corrisponde ad un qualche stato di tensione o di stress, ma non si è ancora in grado di stabilire se tale variazione è provocata da un'emozione piacevole oppure dolorosa.
Nonostante le difficoltà ancora da superare, il fenomeno GSR offre molte possibilità applicative e già le più note case produttrici di microcomputer si stanno contendendo, a suon di milioni di dollari, la licenza di applicare i dispositivi sui propri apparecchi.


ASSOCIATED PRESS

LA MACCHINA DELLA VERITÀ

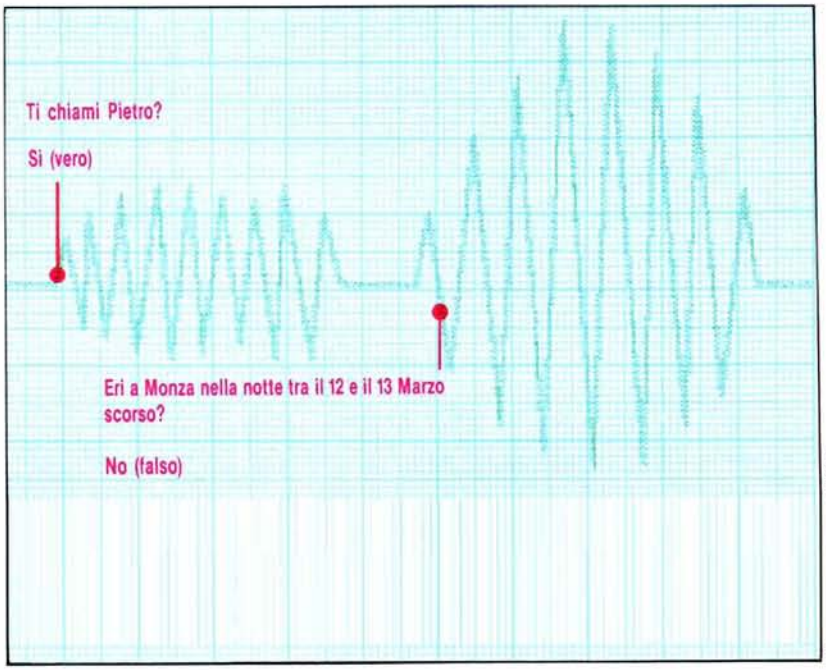
Uno tra i maggiori problemi che la Polizia e gli organi della Giustizia sono costretti ad affrontare è quello di stabilire quando un indiziato o un testimone dice la verità o mente. Sebbene l'esperienza possa insegnare a notare minime variazioni nel comportamento di un individuo (la respirazione, la sudorazione, un improvviso pallore, ecc.), tali valutazioni sono altamente soggettive e non rappresentano che una vaga indicazione. Gli studi in materia hanno condotto alla realizzazione di un apparecchio chiamato "poligrafo" o più comunemente "macchina della verità".
In effetti il dispositivo non è altro che un lettore di resistenza elettrica, che traduce in un grafico (simile agli elettrocardiogrammi o agli elettroencefalogrammi, che funzionano su medesimi principi) le variazioni rilevate. Se un soggetto mente, sul grafico si può notare una sensibile variazione di segnale.
Tuttavia, a distanza di molti anni dalla sua invenzione, l'affidabilità della "macchina della verità" è ancora molto controversa e non ne viene riconosciuto il valore legale di prova in un giudizio

Attitudini ed emozioni

Talvolta, particolarmente negli Stati Uniti, gli aspiranti ad un lavoro di alta responsabilità si offrono volontariamente ad una prova di attitudine svolta con la macchina della verità. Nella foto, un candidato sta esaminando i risultati appena ottenuti in un test attitudinale

Le bugie fanno crescere il... grafico

La cosiddetta "macchina della verità" non fa che misurare le variazioni nella conduttività della pelle dell'interrogato, producendo un grafico del loro andamento. Nella realtà, però, raramente si ottengono grafici così netti e definiti come quello dell'esempio riportato qui sotto: le reazioni emotive che provocano i segnali registrati dall'apparecchio, infatti, possono variare da un individuo all'altro (ciò che causa vergogna o imbarazzo ad uno può non influire su un altro). In realtà l'apparecchio offre soltanto una traccia da interpretare con molta cautela, pertanto non viene quasi mai riconosciuta come elemento determinante nei procedimenti legali



LIZ DIXON

Strutture dinamiche

Le precise modalità necessarie per la gestione di file nel PASCAL

Proseguiamo con lo sviluppo del nostro database (vedere a pag. 1665), stabilendo quali siano le variabili essenziali e realizzando un algoritmo "informale".

```
VAR
    lista :RecordList;
    dimens :Cardinal;
BEGIN
    dimens := 0; {dimensione attiva della lista}
    {legge i dati nella lista, aggiornando la dimensione}
    {ordina la dimensione degli elementi}
    {visualizza le dimensioni degli elmementi}
END.
```

Per convertire questo grossolano algoritmo in qualcosa di più tangibile, occorre esprimere le tre fasi principali sotto forma di chiamate a procedure (di nome appropriato) ed elencare tutti i dati di cui ogni procedura ha bisogno in una lista di parametri. Portata a termine questa semplice operazione, si possono redigere al meglio i vari blocchi delle procedure espresse solo in forma abbozzata. L'ultima frase nel programma, ad esempio, può diventare:

```
Print (lista, dimens)
```

Alla procedura basta passare la lista dei valori e il numero di elementi, così in questo caso nella testata non occorre usare VAR:

```
PROCEDURE Print (elementi : RecordList;
                 high : limiti);
```

Si potrebbe a questo punto codificare la procedura per intero, ma conviene prima dedicarsi alle altre procedure di livello 1, lasciando questa così com'è. È necessario assegnare un nome per ogni procedura, stabilire quali siano i parametri da passare e se qualcuno di questi va passato come parametro VAR (indirizzo).
Tutti i metodi di strutturazione dei dati visti finora hanno una dimensione fissa, specificata nelle definizioni TYPE e pertanto nota al momento della compilazione. Il PASCAL, comunque, offre anche strutture avanzate, che possono variare la propria misura (e, in certi casi, perfino il loro tipo) durante la stessa esecuzione del programma. Le dimensioni di una struttura avanzata sono limitate soltanto dalla memoria (RAM e di massa) disponibile sull'apparecchio. La struttura avanzata più comune è il file sequenziale.
Come in qualsiasi matrice, ogni elemento di un file può essere di qualunque tipo (ossia semplice o strutturato), con una eccezione: non si possono avere file di file. Usando la definizione di tipo adoperata nei record in precedenza, si possono aggiungere le seguenti dichiarazioni:

```
TYPE
    TipoFile = FILE OF dati;
VAR
    FileDati : Tipofile;
```

che ci permettono di creare ed elaborare un file contenente un numero infinito di componenti, ognuno dei quali rappresenta la registrazione di un nominativo, di una quantità numerica e di qualsiasi altro campo occorra. Così come si scrive read(simbolo) per leggere un singolo carattere (char) dal file di immissione, si può anche usare read (FileDati, elem) oppure write (FileDati, elem) e trattare un'intera struttura di record come se fosse un singolo dato. Se detti file occorrono soltanto al momento dell'esecuzione del programma, l'unico accorgimento è quello di usare le apposite procedure predefinite reset e rewrite per aprire, rispettivamente, i file in lettura o in scrittura.
Volendo renderle strutture permanenti (caso molto frequente) allora gli identificatori del file vanno elencati nella testata del programma riservata ai parametri. Nel nostro caso:

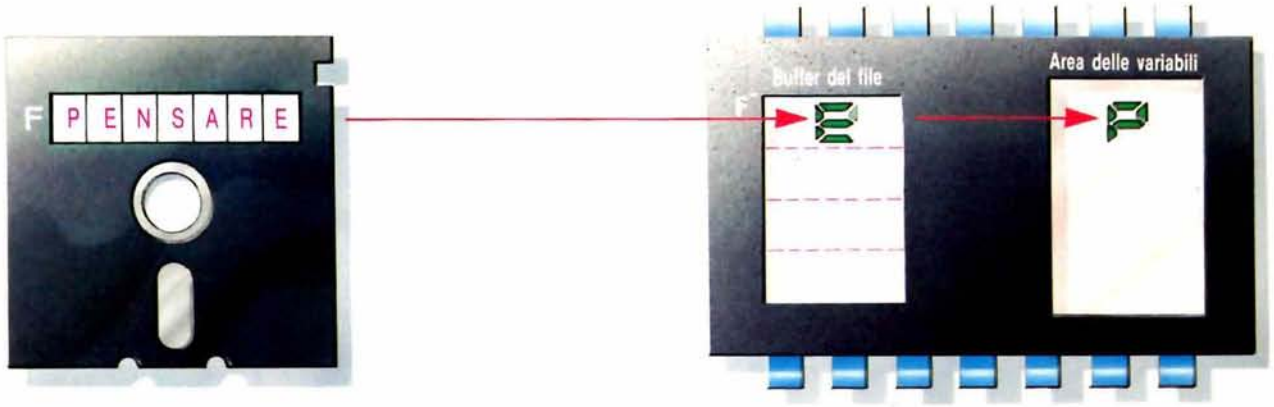
```
PROGRAM TrattamentoDati (input, output,FileDati)
```

Ogni volta che si adoperano le istruzioni read o write, vengono chiamate le procedure predefinite del PASCAL che si occupano di immissione/emissione su file. I due file standard di input e di output, sui microcomputer, sono rispettivamente la tastiera e lo schermo. Considerando questi due dispositivi come normali file, il trattamento dei dati in ingresso e in uscita, in PASCAL, risulta molto omogeneo. Si rammenti, dunque, che quando si scrive write(N), senza indicare nomi di file, il valore di N viene emesso sul normale file di output e, omettendo il nome di file in una read o in una ReadLn, l'immissione avviene estraendo dati dal normale file di input. Entrambi questi file sono di tipo testuale, vale a dire che ogni record del file corrisponde ad un singolo carattere (char).

I file di testo

Diversamente dagli altri file, quelli testuali possono contenere, oltre ad un marcatore di "fine flusso", caratteri speciali che indicano la fine di ogni riga di testo. Il valore di tali marcatori differisce secondo il sistema operativo in uso. Inoltre può essere usato un singolo carattere di controllo, oppure la combinazione di due caratteri (CR-LF, ad esempio). In certi casi, addirittura, viene omesso del tutto, se viene conservata al suo posto la lunghezza in byte della riga.
La funzione EoLn(F) e le due procedure predefi-

LETTURA DA UN FILE



KEVIN JONES

nite ReadLn(F) e WriteLn(F), tutte utilizzabili unicamente con i file testuali, garantiscono una certa compatibilità d'uso tra le varie versioni di PASCAL. L'uso della funzione EoF(F), per rilevare la fine del flusso, è invece esteso a tutti i tipi di file. Poiché non è detto che il marcatore di fine riga (a uno o più caratteri) esista, il valore di un char letto quando la funzione EoLn è vera corrisponde per convenzione a uno "spazio". In ogni caso conviene intercettare l'eventuale fine di riga adoperando prima la funzione EoLn.
I flussi di input e di output esistono prima e dopo l'esecuzione di un programma (sono sempre "aperti") quindi non è necessario localizzarli né crearli specificamente.
Gli altri file, invece, vanno assegnati usando il nome di un dispositivo esterno per poi aprirli in lettura con una chiamata alla procedura reset oppure creare una registrazione nella directory, preparatoria alla scrittura, tramite rewrite. Uno schema generale per il trattamento di un file risulta pertanto molto semplice:

```
{apertura del file}
WHILE {fine del file non raggiunto} DO
    WHILE {fine di riga non raggiunto} DO
    {legge un carattere}
    {elabora il carattere}
    {ignora il marcatore di fine riga}
```

Put e Get

Le procedure read e write vengono in effetti implementate sfruttando i buffer per i file e le funzioni primitive put (per l'output) e get (per l'input). Quando un file viene riscritto, nessun dato è posto nel buffer finché non viene eseguita la relativa istruzione write. Il processo avviene quindi in due fasi:

```
F^ = dato;
put (F)
```

Analogamente, l'istruzione read(F, dato) può essere espressa come:

```
dato := F^
get(F)
```

Pertanto le frasi nella nostra procedura Copia:

```
read (sorgente, carattere);
write (destinazione, carattere)
```

possono essere codificate anche così:

```
destinazione^ = sorgente^
put (destinazione);
get (sorgente)
```

In tal caso non occorre la variabile locale carattere, di tipo char. Una migliore comprensione del funzionamento di ReadLn(F) si può avere esprimendola in termini di funzioni primitive:

```
WHILE NOT EoLn(F) DO
    get(F); {ignora l'eventuale resto della linea}
get (F) {..e ignora il marcatore di fine riga}
```

Quindi, dopo una ReadLn, il buffer del file contiene sempre il primo elemento della riga successiva, che deve essere ancora letta. Tale carattere può essere uno spazio, se è vera EoLn(F), oppure un qualsiasi altro carattere se EoLn(F) è falsa. Ovviamente non è regolare tentare di leggere un file quando è vera la condizione EoF(F), ma è anche errato eseguire qualsiasi operazione, ivi compresa la verifica del buffer del file. Ciò significa che occorre molta attenzione nel trattare file diversi da quelli di input e di output. Aver coscienza di queste limitazioni, tuttavia, favorisce la stesura di programmi "a prova di bomba".
Avendo imparato come si fa a esaminare i caratteri "prossimi venturi" in un particolare flusso, si può formulare la procedura SaltaSpazi, citata in precedenza:

```
PROCEDURE SaltaSpazi (VAR F:testo);
    CONST
        spazio = ' ';
    VAR
        fatto : boolean;
    BEGIN
        fatto := Eof(F);
        IF NOT fatto THEN
```

All'ombra di un file

Nel procedimento di aprire il file F, il PASCAL predispone e abbina ad esso un buffer (area di memoria) F^ in cui viene depositato il primo carattere del flusso. Quando viene eseguita una read, il carattere nel buffer viene assegnato ad una variabile e al suo posto viene depositato il carattere successivo del file. Se le prime lettere del file F sono ad esempio PENSARE, allora all'apertura del file P viene depositato in F^ poi, eseguita la prima lettura con read, la lettera P viene assegnata a una variabile del PASCAL, mentre in F^ finisce il carattere E

```
        fatto := input^ > spazio;
    WHILE NOT fatto DO
        BEGIN
    get(F);
    fatto := EoF(F);
    IF NOT fatto THEN
        fatto := F^ > spazio
    END
END; {SaltaSpazi}
```

Si noti che la procedura salta tutti i caratteri spazio in un testo, oltre a quelli di fine riga. Volendo soltanto saltare uno spazio, la frase condizionale va modificata in:

```
fatto := EoLn(F) OR (F^ > spazio)
```

A questo punto, volendo saltare tutti gli spazi, occorre scrivere:

```
REPEAT
    SaltaSpazi (F);
    IF NOT EoF(F) THEN
        TestoTrovato := NOT EoLn(F)
UNTIL TestoTrovato OR EoF(F)
```

Con questa modifica la stesura di controlli su immissioni nulle è facile. Ad esempio:

```
REPEAT
    write ('Immettere il dato:');
    Saltaspazi (input)
UNTIL NOT EoLn(input)
```

Un simile controllo fallisce però se capita di digitare il carattere di controllo che indica la fine del file, ma con qualche piccolo accorgimento a ciò si può sempre ovviare.
La procedura assign è da molti considerata una estensione standard del PASCAL e può darsi che venga presto inclusa ufficialmente nello standard del linguaggio, assieme a open e seek per l'accesso casuale ai file. Altre estensioni molto comuni sono:

```
FUNCTION Fstat (NomeFile)
```

che produce un risultato booleano (vero se il file esiste già), e:

```
PROCEDURE Rename(VecchioNome, NuovoNome)
```

per cambiar nome ad un file esistente.

Pascal & Pascal

Generalmente in PASCAL non vi è alcun limite al numero di file aperti contemporaneamente. Tuttavia, siccome l'uso dei file passa attraverso il sistema operativo del computer, esistono divergenze nelle varie versioni. Alcune di queste (ad esempio quella della Hisoft), non prevedono addirittura l'uso di file. Inoltre esistono differenze, sempre dovute ai diversi sistemi operativi, circa l'assegnazione dei nomi ai file.
Sui computer più potenti vengono in genere considerati validi i primi 10 caratteri dell'identificatore del file, ma in altri casi esistono limitazioni.
L'estensione standard, per collegare il nome di un file ad un identificatore, è la procedura assign, usata prima di reset o rewrite:

```
assign (IdentificatoreFile, StringaDelNome)
```

Ai fini delle spiegazioni, viene qui usato il formato assign(FileDati,D:NomFil.dat)

PROGRAMMA COPIATESTO

Il PASCAL dispone dell'identificatore 'text' per il tipo più comune di file: questo viene adoperato nella dichiarazione di variabili per file testuali nella testata dei parametri. Qui di seguito è listato un programma per la copia di testi. Per rendere il programma di più generico impiego, la procedura è raccolta in un modulo, chiamato Copia, a sé stante capace di trattare due file testuali qualsiasi.
Si rammenti che compilatori non standard possono richiedere l'impiego di reset (F1,'sorgente') ed espressioni simili nel modulo principale. Si noti che entrambi i file sono passati alla procedura Copia come parametri VAR: ciò perché non solo viene così aggiornato il file di destinazione, ma viene anche letto il file sorgente, il cui stato cambia di continuo. Per questo motivo, le variabili dei file vengono sempre passate per indirizzo e mai per valore. Passare il valore della variabile, infatti, implica creare una copia "locale" con grosso spreco di memoria

```
PROGRAM     CopiaTesto     (F1, F2);
VAR
        F1,
        F2     : text;
{11111111111111111111111111111111}
PROCEDURE     Copia (VAR sorgente,
                          destinazione : text);
    VAR
        carattere : char;
    BEGIN
        WHILE NOT EoF (sorgente) DO
            BEGIN {copia una linea: }
                WHILE NOT EoLn (sorgente) DO
                    BEGIN
                        read (sorgente, carattere);
                        write (destinazione, carattere)
                    END;
                        {Adesso copia il marcatore di fine riga: }
                ReadLn (sorgente);
                WriteLn(destinazione)
            END
END;     {Copia}
{11111111111111111111111111111111}
BEGIN {CopiaTesto — programma principale}
    assign (F1, 'Sorgente');
    reset (F1);     {Localizza e apre per lettura}
    assign (F2, 'Destinazione');
    rewrite (F2);     {Crea il file}
    Copia (F1, F2)
END.
```

Il MicroPen

La descrizione di un buon database per il microcomputer Amstrad CPC 464

Descritto come "un sistema di archiviazione database per il CPC 464", il *MicroPen* è commercializzato dalla Amsoft, una divisione della Amstrad, ma è stato realizzato da una software house indipendente, la Intelligence Ireland. *MicroPen* fa parte di un "trittico" di programmi, composto da un DBM, un elaboratore di testi, e una lavagna elettronica. Il pacchetto di programmi ha diversi punti in comune con il famoso *1-2-3* della Lotus e con il *PIPS* della Sord. Il database manager, o DBM, incorpora un text editor, chiamato *Penform*, utile per creare su schermo le maschere con le quali inserire e richiamare le varie schede. Il modulo del DBM è chiamato semplicemente *Pen*, ossia "penna".

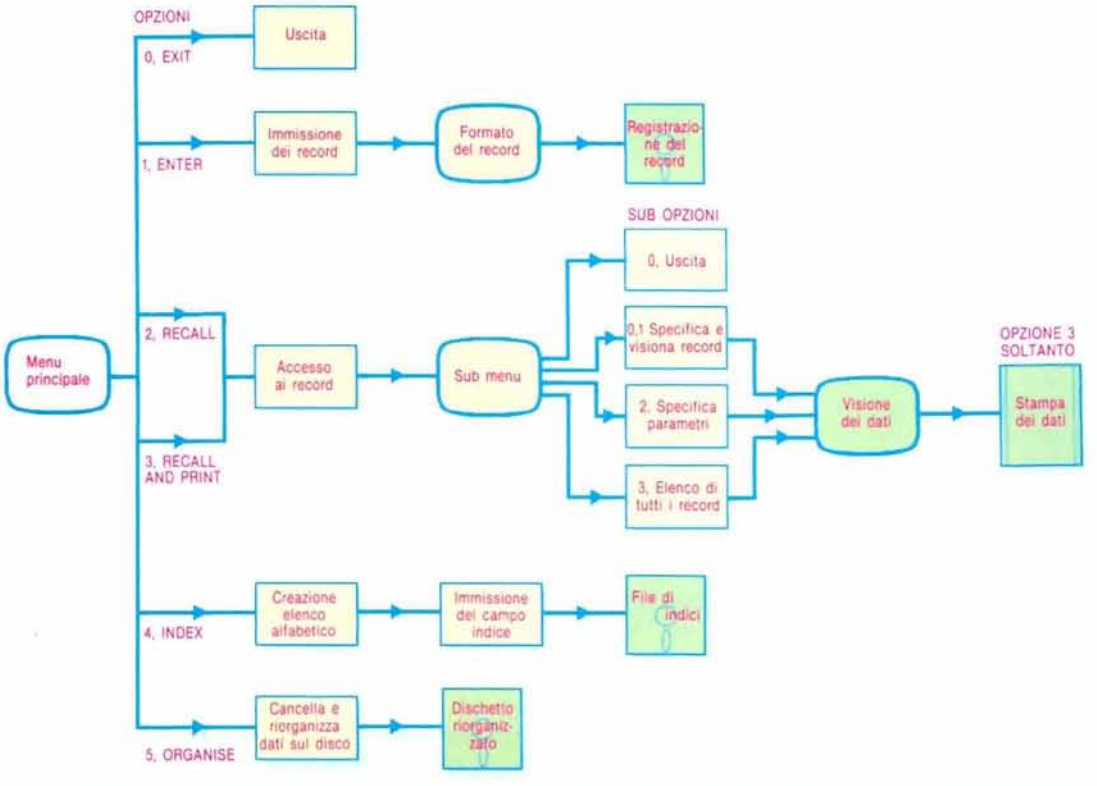
La versione di *MicroPen* per Amstrad gira sotto CP/M-80 e richiede quindi il possesso di un'unità a dischetti DDI-1. Sebbene ciò possa inizialmente sembrare dispendioso, ci si accorge ben presto che l'acquisto dell'unità a dischetti è indispensabile; infatti qualsiasi DBM che operi con un sistema a nastri è lento in modo inaccettabile.

Prima di poter creare un archivio per il database, occorre formulare una maschera per definire il formato delle informazioni da immettere. Il programma da usare a tal scopo è il *Penform*. Questo è un semplice text editor (nel manuale viene adoperato impropriamente il termine 'word processor'), che consente l'immissione di caratteri e simboli sullo schermo, di spostare il cursore e via dicendo. Esso dispone di semplici comandi, quale ad esempio [CTRL]Y per rimuovere una linea o [ESC]E per memorizzare su dischetto la maschera creata.

Esistono alcune limitazioni alla quantità di informazioni che si possono immettere in un file di MicroPen. Un record (anche se il manuale adopera il termine 'layout') può contenere fino a 100 campi, ma in nessun caso più di 1024 caratteri.

Teoricamente un file può contenere fino a 32750 record, ma in realtà un dischetto non possiede una simile capacità: la quantità di memoria necessaria per 32750 record di 124 sarebbe infatti superiore a 33 Megabyte (presupponendo che non vengano adoperate tecniche di compattamento).

Una caratteristica insolita del MicroPen è che, in fase di creazione della maschera, a ciascun campo di un record deve essere attribuito un



Piano d'azione

Queste sono le diverse opzioni e procedure offerte dai menu del "MicroPen". In programmi più sofisticati, quali il "dBaseII", che possiedono linguaggi di programmazione interni, l'utente viene messo in grado di costruirsi le proprie procedure di accesso con molta più elasticità

diverso identificatore. Questi identificatori sono chiamati "marcatori di campo" e possono essere rappresentati con qualsiasi carattere o simbolo visualizzabile, eccezion fatta per le parentesi quadre, che servono da delimitatori.
Nella creazione del solito archivio di magazzino, con il *MicroPen* la maschera potrebbe avere questo aspetto:

```
Numero di Matricola: [A                    ]
Codice di Listino: [B                      ]
Prezzo: [C ] Giacenza: [D                  ]
Descrizione: [E                            ]
Fornitore: [F                              ]
Telefono del fornitore: [G                 ]
Nome dell'addetto: [H                      ]
```

Una volta creata una maschera come questa, essa viene memorizzata su dischetto sotto il nome, ad esempio, PARTI.INP (l'estensione al nome, .INP, è imposta dal programma *MicroPen*).

Una penna che vola

Quando si avvia il programma *Pen*, sullo schermo compare un invito a digitare il nome del database al quale si intende accedere. Volendo usare il nostro archivio di magazzino si immetterebbe il nome PARTI (l'estensione viene assegnata automaticamente come .INP).
Il programma visualizza quindi il menu principale, che offre quattro opzioni:

```
Database:PARTI, per record 328, records on file 0.

0 = Exit,  1 = Enter,  2 = Recall,  3 = Recall&Print,  4 = Index,
5 = Organise
```

L'Opzione 1 serve per l'immissione dei record. La riga superiore comunica dati 'statistici': ogni record è composto da 328 caratteri e non vi sono (almeno per il momento) record nell'archivio.
Selezionando l'Opzione 1 sullo schermo compare la maschera e le istruzioni per l'uso. Nel nostro caso:

```
Press ESC when data entry completed for record 1
Press to clear field
Numero di Matricola:
Codice di Listino:
Prezzo: Giacenza:
Descrizione:
Fornitore:
Telefono del fornitore:
Nome dell'addetto:
```

I dati vengono immessi nel modo consueto. La pressione di Return termina l'immissione di un campo e sposta il cursore sul campo successivo.
Errori di digitazione, finché si è all'interno di un campo, si possono eliminare premendo il tasto Delete.
Terminata l'immissione del record, compare un menu secondario:

```
0 = Exit
1 = Continue
2 = Write Record to File
```

L'Opzione 2 scrive il record sul dischetto, prima di ripresentare la maschera d'immissione.
L'Opzione 2 del menu principale, invece, consente l'accesso e il richiamo di record. Prima però compare un altro menu secondario:

```
Recall by: 0 = Exit
           1 = Record number
           2 = Search
           3 = List all on file
```

L'Opzione 0 serve per uscire dal programma. L'Opzione 1, richiamo in base al numero di record, è utile soltanto se tale numero è noto, il che non è sempre vero. L'Opzione 2 consente di fissare alcuni parametri di ricerca, per trovare tutti i record che contengono particolari caratteristiche.
Il relativo sotto-menu è:

```
Press Escape when search profile complete. To set
search mode:
^Q = Contains, ^W = Not contains, ^E = Equals, ^R = Not
equals, ^Q = Greater, ^Y = Less
```

Il formato dei comandi è caratterizzato da una certa incoerenza: talvolta basta il tasto ESC, altre volte occorre il CTRL + lettera, altre ancora basta un numero. Ciò non è giustificato da esigenze mnemoniche dei comandi, ma pare invece frutto di una scelta puramente arbitraria.
I comandi di questo sotto-menu permettono di visualizzare i record che contengono la particolare informazione specificata oppure quelli che non la contengono o, ancora, quelli in cui un particolare campo coincide, è maggiore o è minore rispetto a quello specificato. La ricerca, ad esempio, di tutti gli articoli di magazzino il cui PREZZO è inferiore alle 9700 è molto facile.
Micropen offre anche la possibilità di creare un indice di tutti i record memorizzati nell'archivio. Per far ciò occorre specificare il campo da usare come indice e conoscere il numero totale dei record. Per specificare il campo, va usato l'identificatore e non il nome ad esso attribuito nella maschera. Per ottenere un indice in base al codice di listino, avendo 500 record nel database, si digita il comando B = #500 (B è l'identificatore del secondo campo della maschera).
Concludendo, *MicroPen* è un package che offre, ad un prezzo molto ragionevole, caratteristiche di ricerca e di creazione di indici niente affatto disprezzabili. Non possiede, al pari di *Archive* o del *dBasell*, un linguaggio interno di programmazione: se da un lato il *dBasell* è un package decisamente costoso, dall'altro *Archive* viene dato "gratis" assieme al QL della Sinclair, anche se questo apparecchio costa quanto un Amstrad CPC 464 dotato di unità a dischetti e del software per il *MicroPen*.

Il DDI-1 della Amstrad

Un accessorio indispensabile: l'unità a dischetti per il computer Amstrad

In molte case gli home computer sono ormai divenuti parte dell'arredamento: alla stessa stregua di altri elettrodomestici, come la televisione o l'impianto ad alta fedeltà, trovano la loro stabile collocazione, magari nel salotto di casa.
L'Amstrad è un apparecchio che ha ottenuto molto successo tra gli appassionati dell'informatica "casalinga", ma la mancanza di un'unità a dischetti (nonostante la presenza di un discreto registratore incorporato) ha influito negativamente sulla sua diffusione.
Finalmente, però, l'unità a dischetti, preannunciata con molto anticipo, ha raggiunto il mercato.
Il package comprende l'unità a dischetti, un manuale, il dischetto di sistema e l'interfaccia per il collegamento all'apposita porta del computer. L'altro capo del cavo piatto a 34 poli viene inserito nel connettore che si trova sul lato posteriore del disk drive.
Il formato dei dischetti, da 3 pollici, risponde allo standard Hitachi. Questa può sembrare una scelta discutibile, dato che il formato prevalente è quello della Sony, da 3 1/2 pollici, adoperato da molte altre Case conosciutissime (la Apple, la Apricot e, recentemente, la Acorn per l'Electron).
Oltre alla Amstrad, nessun altro fabbricante di computer sembra aver optato per il formato Hitachi.
I dischetti usati dall'Amstrad sono costruiti con i medesimi criteri impiegati nei dischetti Sony, ma il loro aspetto è alquanto diverso: le dimensioni sono di mm. 100 x 80 x 4, ma come i Sony sono racchiusi in una custodia plastica e la finestrella di lettura/scrittura è protetta contro la polvere e poco opportune ditate da una lamina metallica, che scorre quando il dischetto viene infilato nel drive.
All'interno dell'unità si trovano due motori, uno dei quali controlla la rotazione del dischetto mentre l'altro provvede agli spostamenti della testina magnetica di lettura/scrittura. La sezione di alimentazione, interna all'unità e opportunamente schermata con una lamina di metallo, si trova nella parte posteriore dell'apparecchio. La schermatura ha la duplice funzione di dissipare il calore prodotto dal funzionamento e di proteggere i dischetti dal campo magnetico generato dal trasformatore. L'interruttore di accensione è collocato sul lato posteriore dell'unità.
L'apparecchio funziona subito bene ed è piuttosto affidabile: non si hanno problemi a rintracciare i file, che vengono caricati in memoria in un paio di secondi. Il rumore prodotto durante il funzionamento è forse superiore a quello delle unità di tipo Sony, ma certamente inferiore a quello prodotto dalla maggior parte delle unità da 5 1/4 pollici.



CHRIS STEVENS

Un passo verso l'alto
L'unità a dischetti Amstrad DDI-1 consente di ampliare le prestazioni del proprio computer, elevando il sistema ad un livello paragonabile a quello degli apparecchi per applicazioni commerciali. Infatti, grazie anche al gruppo di programmi forniti assieme all'unità (il famoso sistema operativo CP/M, l'AmsDOS e il DrLOGO), l'Amstrad CPC 464 estende notevolmente il proprio campo applicativo

Il dischetto di sistema

Sul dischetto di sistema si trovano tre programmi di utilità. Uno è l'*AmsDOS*, il sistema operativo proprio della Amstrad. Gli altri due provengono dalla Digital Research e sono il notissimo sistema operativo per dischi CP/M e il DrLOGO, una versione alquanto popolare della grafica della tartaruga, che sta gradualmente soppiantando l'ormai vecchio BASIC.
Il sistema operativo AmsDOS è forse il pezzo meno forte del pacchetto. Inoltre, per caricarlo in memoria, occorre prima leggere il CP/M. Questo è un procedimento un po' laborioso, comunque l'AmsDOS non sembra occupare una maggiore quantità di memoria, quindi è certo che vada a sovrapporsi a parte del CP/M. Diversamente da questo, che è un sistema operativo a sé stante, l'AmsDOS si limita ad aggiungere i comandi per dischetti, che vanno preceduti dal simbolo di barra | (*Shift* @). Per esempio: |DRIVE e |DIR.

Il metodo seguito dall'AmsDOS per il trattamento dei flussi è quanto di più peculiare si sia mai visto su di un home computer. Ad esempio, per cancellare un file (comando ERA), occorre prima assegnare il nome del file a una stringa e poi impartire il comando di cancellazione. Supponiamo di voler cancellare il file FATTURE.$$$. La sequenza di comandi è:

```
A$ = "FATTURE.$$$"
| ERA,a$
```

Le difficoltà si avvertono ancor di più se l'operazione è quella di cambiar nome a un file (comando REN); infatti il vecchio e il nuovo nome vanno assegnati a due diverse stringhe prima di poter impartire il comando voluto. Molti utenti preferiscono caricare il CP/M per eseguire questo genere di operazioni, in quanto il formato dei comandi è decisamente più semplice. Un altro inconveniente è l'impossibilità di usare, sotto CP/M, il BASIC della Amstrad: si è obbligati all'impiego dell'AmsDOS se si desidera programmare in tale linguaggio.
Il CP/M fornito sul dischetto di sistema è nella versione 2.2, adottata durante l'ultimo decennio su innumerevoli microcomputer a 8 bit per applicazioni commerciali. Sul dischetto sono presenti anche tutti i vari comandi CP/M cosiddetti "transienti", ossia caricati in memoria soltanto quando occorre e rimossi automaticamente quando non servono più. Il guaio è che diversi di questi comandi (ad esempio il PIP, che trasferisce i file da una periferica all'altra) presuppongono l'esistenza di due unità a dischetti, quindi risultano di scarsa applicabilità.
Per ovviare alla mancanza del secondo drive, la Amstrad ha ritenuto opportuno aggiungere un nuovo comando transiente per ottenere il trasferimento di file da un dischetto all'altro. Il comando, chiamato FILECOPY, fa sì che appaia un messaggio per avvertire quando è il momento di scambiare il dischetto "sorgente" con quello di "destinazione" e viceversa. Per la copia di interi dischetti si usa un altro comando transiente prodotto dalla Amstrad: DISCCOPY.

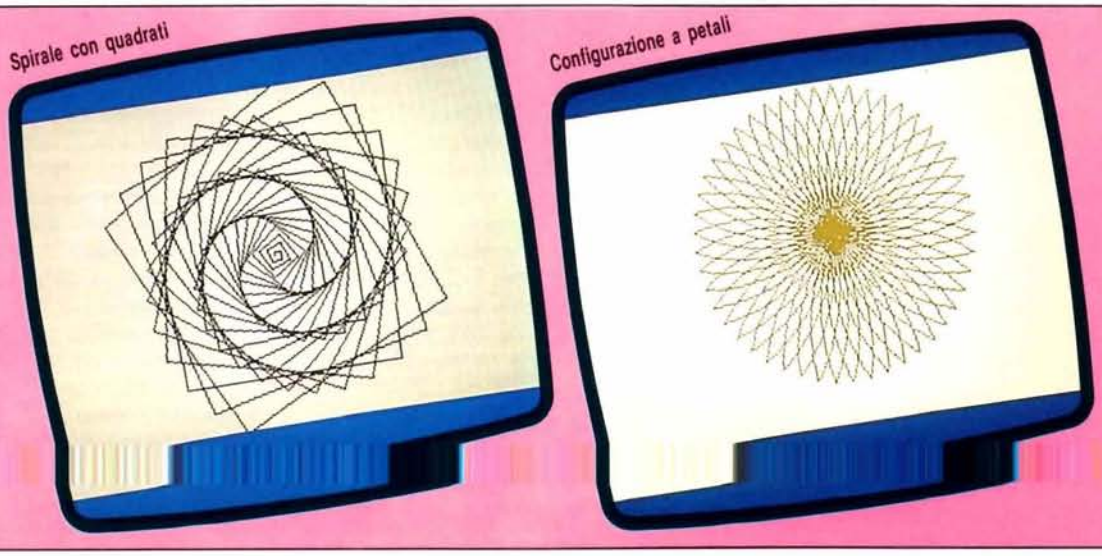
In definitiva, per sfruttare appieno la potenzialità del computer e dei due sistemi operativi (specialmente il CP/M) è indispensabile l'acquisto di una seconda unità a dischetti.
Se venisse da chiedersi come mai la Amstrad abbia voluto offrire un sistema operativo come il CP/M, che sembra stare stretto nella configurazione di base del computer, una probabile risposta va cercata nella strategia a lungo termine della Casa produttrice. La politica della Amstrad, infatti, non sembra mirare alla realizzazione di prodotti di punta, quanto alla commercializzazione di apparecchi affidabili e ben sperimentati, adatti ad una vasta gamma di applicazioni. Solo così si spiega la scelta di adottare il chip Z80 a 8 bit, quando tutti gli altri fabbricanti si accaniscono nella produzione di computer a 16 bit. L'adozione del CP/M è quindi una scelta consequenziale quasi scontata, sebbene non si tratti di un sistema operativo particolarmente comodo da usare (nonostante il vastissimo successo). Resta il fatto che il CP/M è ancora adoperato da decine di migliaia di programmatori in tutto il mondo e sono disponibili innumerevoli programmi già pronti e di sicuro funzionamento.
A parte le citate limitazioni nell'impiego di una sola unità a dischetti, la Amstrad va senz'altro elogiata per il fatto di offrire una completa implementazione del sistema operativo CP/M su di un home computer: solo pochi anni fa gli apparecchi dotati di CP/M avevano un prezzo decisamente troppo alto per i semplici appassionati.

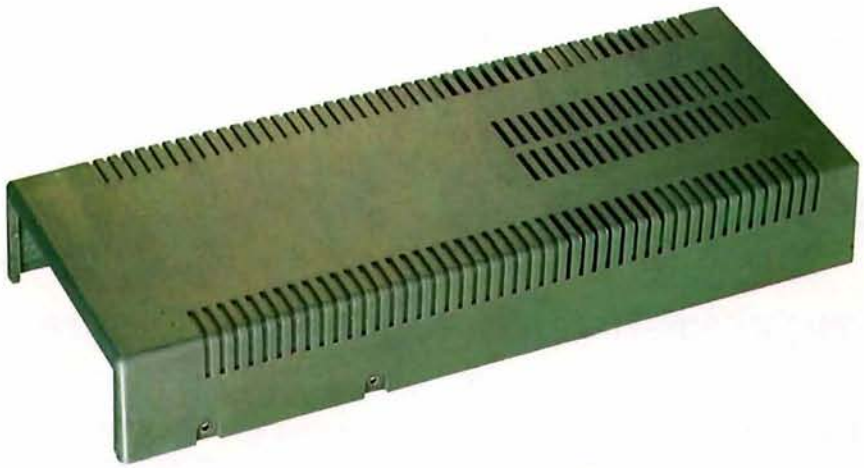
Il LOGO della Digital Research

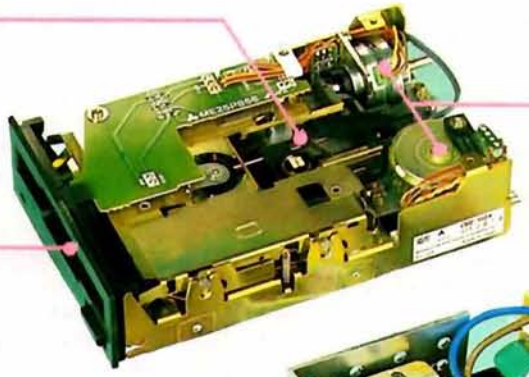
Il DrLOGO è una delle tante versioni del noto linguaggio LOGO, largamente adoperato nel settore didattico. Il DrLOGO, che gira sotto CP/M, ricorre al sistema operativo per tutte le funzioni di accesso ai dischetti, quindi il CP/M va caricato in memoria prima di poter usare il LOGO.
Il linguaggio consente l'accesso alle funzioni grafiche e sonore dell'Amstrad, secondo le

TARTARUGA AL LAVORO

Questi disegni sono stati realizzati con brevissimi programmi scritti in DrLOGO, la versione di LOGO prodotta dalla Digital Research. Il DrLOGO non consente di modificare la sagoma della tartaruga sullo schermo, ma in compenso è in grado di accedere alle funzioni audio dell'apparecchio







Testina di lettura/scrittura
Appoggiandosi leggermente sulla superficie del dischetto, la testina intercetta o modifica i segnali magnetici

Motori
Un motore controlla la rotazione del dischetto, l'altro si occupa di posizionare la testina di lettura/scrittura

Fessura di inserimento
I microdischetti da 3 pollici, formato Hitachi, vengono inseriti in questa fessura



Alimentatore
Il DDI-1 è dotato di una sezione di alimentazione autonoma adeguatamente schermata per evitare che il campo magnetico generato disturbi i segnali passanti attraverso la testina di lettura/scrittura o quelli registrati sul dischetto

Scheda del circuito
Questa scheda regola la tensione di alimentazione in modo da pilotare i motori elettrici a una velocità regolare

CHRIS STEVENS

AMSTRAD DDI-1

DIMENSIONI

mm. 280 x 105 x 75

CAPACITÀ

Dischi di sistema: 169 Kbyte
Dischi di dati: 178 Kbyte

INTERFACCE

Cavo piatto a 34 poli (segnali paralleli) da collegare alla porta del computer. Il cavo consente l'uso di due unità a dischetti

DOCUMENTAZIONE

Il manuale è abbastanza buono, ma talvolta è difficile individuare le informazioni che possono interessare

PRO

La disponibilità del CP/M e del DrLOGO amplia notevolmente la gamma di applicazioni del computer

CONTRO

L'AmsDOS presenta qualche inconveniente e non è facilissimo da usare. Sia adoperando il CP/M che l'AmsDOS con una sola unità a dischetti si avverte la mancanza di una seconda unità, indispensabile per sfruttare appieno i due sistemi operativi

modalità note a chi ha seguito la sezione dedicata al LOGO (che inizia a pagina 986). Inoltre esiste la possibilità di operare con il joystick, ampliando così la gamma di giochi che si possono realizzare in LOGO.
Sebbene l'implementazione del DrLOGO sia molto buona e regga bene il confronto con tante altre versioni del linguaggio, possono sorgere dei dubbi sul fatto che essa risenta di qualche limitazione dovuta all'hardware. Infatti, una prova eseguita con un programma di tassellazione (la ripetuta riproduzione di disegno geometrico fino a riempire l'intero schermo), perfettamente funzionante su di un Commodore 64, ha prodotto un messaggio di «spazio insufficiente nello stack» sull'Amstrad. Ciò può indicare che forse il linguaggio sta un po' stretto nella memoria di questo computer. A parte ciò, il DrLOGO è veloce e facile da usare e rappresenta un'ottima introduzione al linguaggio. Come sempre, a decretare il successo o il fallimento di un apparecchio è la disponibilità di software.
La Amstrad ha realizzato una serie di programmi di utilità da usare con l'unità a dischetti, molti dei quali sembrano orientati ad applicazioni cosiddette "serie". Fra questi programmi c'è un database, un elaboratore di testi e anche una versione del linguaggio Pascal.
Tirando le somme, con il lancio dell'unità a dischetti la Amstrad offre un sistema che non limita più l'impiego del computer da essa prodotto al solo settore dei giochi. L'unità a dischetti DDI-1 ha un prezzo molto competitivo e può invogliare molti appassionati che desiderano apprezzare per la prima volta le capacità di un computer dotato di una memoria di massa molto più maneggevole e rapida rispetto ai nastri magnetici.

Serrare i portelli di boccaporto

Le routine per i restanti quattro eventi che possono capitare nel viaggio

Nel programma, la scelta degli eventi viene fatta ricorrendo alla subroutine che inizia alla linea 6500. Questa genera un numero casuale, utilizzato poi alla linea 6510 nella frase ON X GOSUB. Tutto ciò è stato descritto in dettaglio a pag. 1672. Per aggiungere gli ulteriori quattro eventi riportati qui di seguito, la linea 6510 va modificata in:

```
6510 ON X GOSUB 6530,6700,6800,6900,7000,7050
```

Al terzo numero di linea corrisponde una subroutine che tratta l'attacco alla nave da parte di pirati. Prima di tutto il programma verifica se l'evento è già accaduto, esaminando il flag dell'evento per questa subroutine, ossia M(3). Se M(3) vale 1, l'esecuzione torna al programma principale, altrimenti prosegue nella subroutine, ponendo il flag a 1 (linea 6818) per evitare future ripetizioni.
Può darsi che tutto l'equipaggio sia deceduto e in tal caso l'attacco dei pirati va evitato: un ciclo controlla ciascun elemento della matrice dell'equipaggio e conta tutti i membri che hanno una forza di valore 0 o −999. Il conteggio avviene usando la variabile X e se, al termine, X vale 16 significa che non vi è alcun membro vivo, cosicché il controllo ritorna al programma principale.
Quindi, se l'evento non è già accaduto e almeno parte dell'equipaggio è viva, i pirati muovono il loro attacco, che può provocare la morte di qualche membro. Ciò dipende dal fatto che la ciurma abbia o no armi per difendersi. Un conto dei decessi viene tenuto in K, il cui valore minimo è due. La linea 6825 assegna il valore iniziale a K, poi il programma esamina l'appropriato elemento della matrice delle scorte, OA(2), per vedere se ci sono fucili a bordo. Se il valore di OA(2) è 0 o −999 non ci sono armi: K viene posta a 4 e a S$ viene assegnata la stringa NON CI SONO FUCILI. Se invece le armi ci sono, nella variabile S$ viene posta la frase NONOSTANTE I FUCILI.
Un altro ciclo, tra le linee 6836 e 6845, provvede ad eliminare l'appropriato numero di membri dell'equipaggio. La linea 6835 azzera la variabile X per tenere il conto. Se il "rendimento" di un particolare elemento vale 0 o -999, il ciclo passa all'elemento successivo, altrimenti la linea 6840 modifica il valore del rendimento in −999 e incrementa di 1 il valore di X. Quando la X assume un valore pari al numero di marinai uccisi (conservato in K), la linea 6842 provvede a modificare il valore del contatore T, portandolo a 16, in modo da uscire dal ciclo. Se il numero dei vivi è inferiore a K il programma esce naturalmente dal ciclo quando T raggiunge il valore 16. A questo punto viene visualizzato il numero di morti, per completare la frase iniziata alla linea 6828 o alla 6830, e viene chiesto di premere un tasto qualsiasi.

Danni al timone

L'imprevisto successivo riguarda il timone, che può danneggiarsi e richiedere riparazioni. Se a bordo c'è un meccanico (vivo, ovviamente) la riparazione viene fatta in breve tempo e il viaggio può riprendere tranquillamente. Se invece il meccanico non c'è il viaggio subisce un forte ritardo. La subroutine per i danni al timone inizia alla linea 6900 (quarto numero di linea nella ON X GOSUB). Di nuovo, viene eseguito un controllo per vedere se l'evento si è già presentato, esaminando stavolta se l'elemento M(4) vale 1 oppure no.
Alla variabile X viene assegnato il valore 4, il numero di settimane di ritardo se non c'è un meccanico a bordo. Il programma cerca quindi in TS(,) la presenza di un meccanico vivo. Di questo si occupa un ciclo tra le linee 6930 e 6938: in TS(,) si cerca il numero 3, che corrisponde al meccanico, e poi viene controllato che il suo rendimento non sia 0 o −999. Se tutto va bene, il valore di X viene diminuito a 1, in quanto il meccanico riesce a riparare il danno in breve tempo e il viaggio subisce una sola settimana di ritardo.
Sullo schermo viene visualizzato il messaggio parziale NONOSTANTE IL MECCANICO (oppure, se X vale 4, NIENTE MECCANICI E), seguito dal numero di settimane di ritardo nel viaggio. Il valore viene poi sommato alla lunghezza totale del viaggio, EW, alla linea 6965. Premendo qualsiasi tasto si prosegue nel gioco. La variabile X ha qui due funzioni: indica il numero di settimane di ritardo e inoltre funge anche da flag per indicare se il meccanico c'è o no.
Una tempesta è il quinto evento notevole che può capitare e la relativa subroutine inizia alla linea 7000. L'effetto principale è quello di far deviare la nave dalla sua rotta, allungando così il tempo della traversata. Se tra i membri dell'equipaggio c'è un navigatore, la nave viene presto riportata in rotta, altrimen-

IAN McKINNELL

ti il viaggio subisce un forte ritardo. La routine è molto simile a quella che si occupa dei danni al timone tanto che, in parte, utilizza lo stesso codice.
Il programma controlla se l'evento è già capitato, esaminando se l'elemento M(5) vale 1. L'eventuale ritardo massimo, ossia 2 settimane se non c'è un navigatore a bordo, è depositato in X. Un ciclo, tra le linee 7030 e 7038, cerca la presenza del navigatore (tipo numero 4) e, se lo trova, controlla anche che sia vivo. Se tutto va bene il valore di X viene portato a 1, T a 16 e si esce dal ciclo.
In S$ viene depositato il messaggio parziale NONOSTANTE IL NAVIGATORE oppure, se X vale 4, NON CI SONO NAVIGATORI E. Il programma passa poi alla linea 6950, riutilizzando così la parte della subroutine del timone che visualizza il ritardo subito dal viaggio, prima di tornare al programma principale.

Isola a babordo!

L'ultimo grande evento è l'avvistamento di un'isola: la relativa subroutine inizia alla linea 7050 (ultimo numero di linea nella ON X GOSUB). L'isola non si trova esattamente sulla rotta di navigazione, cosicché approdarvi significa allungare la durata del viaggio. D'altro canto, una sosta sull'isola offre la possibilità di fare alcuni rifornimenti. Il giocatore deve quindi decidere se proseguire nel viaggio o se deviare e fermarsi sull'isola. Nel secondo caso, la possibilità di rifornirsi di acqua e viveri dipende in larga misura da eventuali incontri precedenti con l'albatro: se questo è stato abbattuto la spedizione sull'isola non produce risultati.
Come al solito, per prima cosa si verifica se l'evento è già capitato. In caso negativo il giocatore viene informato dell'avvistamento e avvisato di vantaggi e svantaggi esistenti; gli viene quindi chiesto se intende approdare oppure no. La linea 7082 attende una risposta e ne esamina il primo carattere. Se la risposta è negativa, il controllo torna al programma principale tramite la linea 7086. Se invece la risposta è SI o S, la linea 7100 avverte che l'isola è stata raggiunta.
Il programma controlla quindi se il giocatore ha sparato all'albatro oppure no. Se l'uccello non è stato abbattuto (la variabile B$, posta a N alla linea 48, non ha cambiato valore) l'esecuzione passa alla linea 7110. Se, al contrario, il giocatore ha sparato all'albatro, allora la linea 6162 ha assegnato a B$ il valore S e in tal caso l'acqua dell'isola risulta avvelenata e non esiste niente di commestibile. Il programma passa poi alla linea 7130, che aumenta la durata del viaggio.
Se l'uccello non è stato abbattuto, un ciclo che conta da 1 a 4 (tanti sono i tipi di scorte extra) viene aperto alla linea 7110. La quantità di derrate è decisa in modo casuale. La linea 7112 genera un numero casuale e se questo è minore di .25 passa alla scorta successiva. Ciò significa che vi è una probabilità su quattro di ottenere un particolare tipo di scorta. La linea 7115 genera un numero compreso tra 5 e 14, depositandolo in X. Tale valore è visualizzato dalla linea 7120, seguito dall'appropriata unità di misura della scorta (kg o barili). Qualora un precedente evento avesse fatto esaurire una scorta, il corrispondente valore nella matrice sarebbe −999, così la linea 7122 lo pone a zero, prima di registrare la quantità appena rifornita. La linea 7125 si occupa di sommare la quantità a quella già esistente. La sosta all'isola allunga il viaggio di una o due settimane, valore deciso casualmente dalla linea 7135 e sommato alla durata totale dalla linea 7140.
Nella prossima parte viene preso in esame l'ammutinamento dell'equipaggio.



MODULO 9: ALTRI EVENTI PRINCIPALI

Inizializzazione di flag

```
48 A$="N":B$="N"
```

Routine dei pirati

```
6800 REM ARRIVANO I PIRATI!!
6805 IF M(3)=1 THEN RETURN
6810 X=0
6812 FOR T=1 TO 16
6814 IF TS(T,2)=0 OR TS(T,2)=-999 THEN
     X=X+1
6815 NEXT
6816 IF X=16 THEN RETURN
6818 M(3)=1
6820 PRINT CHR$(147)
6822 S$ =" I PIRATI ATTACCANO LA NAVE!*":
     GOSUB 9100
6824 PRINT: GOSUB 9200
6825 K=2
6826 IF OA(2)=0 OR OA(2)=-999 THEN K=4
6828 S$ = "NONOSTANTE LE ARMI*"
6830 IF K=4 THEN S$ = "NON CI SONO ARMI*"
6832 GOSUB 9100
6835 X=0
6836 FOR T=1 TO 16
6838 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 6845
6840 X=X+1: TS(T,2)=-999
6842 IF X=K THEN T=16
6845 NEXT
6850 PRINT X;
6855 S$ = "MEMBRO RIMANE UCCISO*"
6856 IF X>1 THEN S$="MEMBRI RIMANGONO
     UCCISI*"
6860 GOSUB 9100
6865 PRINT: GOSUB 9200
6890 S$=K$: GOSUB 9100
6895 GET I$:IF I$="" THEN 6895
6899 RETURN
```

Routine del timone

```
6900 REM GUAI AL TIMONE!
6905 IF M(4)=1 THEN RETURN
6910 PRINT CHR$(147)
6915 M(4)=1
6920 S$ = "PROBLEMI AL TIMONE!*":GOSUB 9100
6925 PRINT: GOSUB 9200
6928 X=4
6930 FOR T=1 TO 16
6935 IF TS(T,1)=3 AND TS(T,2)<>0 AND TS(T,2)
     <>-999 THEN X=1: T=16
6938 NEXT
6940 S$ = "ANCHE SE C'E' IL MECCANICO*"
6945 IF X=4 THEN S$ = "NON C'E' IL MECCANICO
     E*"
6950 GOSUB 9100
6955 S$ = "IL VIAGGIO SI ALLUNGA DI*":GOSUB
     9100
6960 PRINT X;"SETTIMANE"
6965 EW=EW+X
6967 PRINT: GOSUB 9200
6969 S$=K$: GOSUB 9100
6970 GET I$: IFI$="" THEN 6970
6975 RETURN
```

Routine della tempesta

```
7000 REM TEMPESTA
7005 IF M(5)=1 THEN RETURN
7010 PRINT CHR$(147)
7015 M(5)=1
7020 S$ = "UNA TEMPESTA DEVIA *":GOSUB 9100
7022 S$ = "LA ROTTA DELLA NAVE*":GOSUB 9100
7025 PRINT: GOSUB 9200
7028 X=2
7030 FOR T=1 TO 16
7035 IF TS(T,1)=4 AND TS(T,2)<>0 AND TS(T,2)
     <>-999 THEN X=1: T=16
7038 NEXT
7040 S$ = "ANCHE SE C'E' IL NAVIGATORE*"
7045 IF X=2 THEN S$ = "NON C'E' IL
     NAVIGATORE E*"
7049 GOTO 6950
```

Routine per visitare l'isola

```
7050 REM ISOLA IN VISTA
7055 IF M(6)=1 THEN RETURN
7060 PRINT CHR$(147)
7065 M(6)=1
7070 S$ = "LE CARTE MOSTRANO LA *":GOSUB
     9100
7071 S$ = "PRESENZA DI UN'ISOLA*":GOSUB
     9100
7072 S$ = "SULLA QUALE RIFORNIRSI DI*":
     GOSUB 9100
7073 S$ = "SCORTE. APPRODARE PERO'*":
     GOSUB 9100
7074 S$ = "FA PERDERE TEMPO*":GOSUB 9100
7075 PRINT:GOSUB 9200
7080 S$ = "DESIDERI APPRODARE (S/N)*":
     GOSUB 9100
7082 INPUT I$:I$=LEFT$(I$,1)
7084 IF I$<>"S" AND I$<>"N" THEN 7082
7086 PRINT: GOSUB9200
7090 IF I$="N" THEN 7145
7100 S$="L'ISOLA E' RAGGIUNTA*":GOSUB 9100
7105 S$="SCORTE RACCOLTE:*":GOSUB 9100
7106 IF B$="N" THEN 7110
7107 PRINT: GOSUB 9200
7108 PRINT "NESSUNA!!!":GOSUB 9200
7109 S$="(RICORDA L'ALBATRO!)":GOSUB 9100:
     GOTO 7130
7110 FOR T=1 TO 4
7112 IF RND(1)<.25 THEN 7129
7115 X=INT(RND(1)*10)+5
7120 PRINTX;U$(T);"  DI"; P$(T)
7122 IF PA(T)=-999 THEN PA(T)=0
7125 PA(T)=PA(T)+X
7129 NEXT
7130 S$="ADESSO IL VIAGGIO RICHIEDE*":GOSUB
     9100
7135 X=INT(RND(1)*2)+1
7139 PRINT X;:S$="SETTIMANE IN PIU'*":GOSUB
     9100
7140 EW=EW+X
7145 PRINT:GOSUB 9200
7150 S$=K$:GOSUB 9100
7155 GET I$:IF I$="" THEN 7155
7159 RETURN
```

Basic & Basic

Spectrum:

Apportare le seguenti modifiche:

```
6512 IF X=3 THEN GOSUB 6800
6513 IF X=4 THEN GOSUB 6900
6514 IF X=5 THEN GOSUB 7000
6515 IF X=6 THEN GOSUB 7050

6820   CLS
6895   LET I$=INKEY$:IF I$=""THEN GO TO 6895
6910   CLS
6970   LET I$=INKEY$: IF I$=""THEN GO TO 6970
7010   CLS
7060   CLS
7082   INPUT I$: LET I$(1 TO 1)
7155   LET I$=INKEY$: IF I$=""THEN GO TO 7155
```

BBC Micro

Apportare le seguenti modifiche:

```
6820   CLS
6895   I$=GET$
6910   CLS
6970   I$=GET$
7010   CLS
7060   CLS
7155   I$=GET$
```

DATABASE

La quinta tabella della mappa di memoria del Commodore 64. Per concessione della Commodore Business Machines

INDIRIZZO ESADECIMALE	ETICHETTA	LOCAZIONE DECIMALE	DESCRIZIONE
00A7	INBIT	167	**Rs-232 Input Bits / Cassette Temp** Porta seriale RS-232 Bit in ingresso / Registratore
00A8	BITCI	168	**RS-232 Input Bit Count / Cassette Temp** Porta seriale RS-232 Contatore dei bit in ingresso / Registratore
00A9	RINONE	169	**RS-232 Flag: Check for Start Bit** Porta seriale RS-232: Flag per verifica sul bit di Start
00AA	RIDATA	170	**Rs-232 Input Byte Buffer / Cassette Temp** Porta seriale RS-232: Buffer per il byte in ingresso / Registratore
00AB	RIPRTY	171	**RS-232 Input Parity / Cassette Short Cnt** Porta seriale RS-232: Parità in ingresso / contatore breve per registratore
00AC-00AD	SAL	172-173	**Pointer: Tape Buffer / Screen Scrolling** Puntatore al buffer del registratore / Scorrimento dello schermo
00AE-00AF	EAL	174-175	**Tape End Addresses/ End of Program** Indirizzo di fine nastro / Fine del programma
00B0-00B1	CMPO	176-177	**Tape Timing Constants** Costanti di temporizzazione per il registratore
00B2-00B3	TAPE1	178-179	**Pointer: Start of Tape Buffer** Puntatore all'inizio del buffer per registratore
00B4	BITTS	180	**RS-232 Out Bit Count / Cassette Temp** Porta seriale RS-232: Contatore di bit in uscita / Registratore
00B5	NXTBIT	181	**RS-232 Next Bit to Sent / Tape EOT Flag** Porta seriale RS-232: Prossimo bit da emettere / Flag di Fine Testo per il registratore
00B6	RODATA	182	**RS-232 Out Byte Buffer** Porta seriale RS-232: Buffer per il byte in uscita
00B7	FNLEN	183	**Lenght of Current File Name** Lunghezza del corrente nome di file
00B8	LA	184	**Current Logical File Number** Numero logico assegnato al file corrente
00B9	SA	185	**Current Secondary Address** Indirizzo secondario corrente
00BA	FA	186	**Current Device Number** Numero del corrente dispositivo
00BB-00BC	FNADR	187-188	**Pointer: Current File Name** Puntatore al nome del corrente file
00BD	ROPRTY	189	**RS-232 Out Parity / Cassette Temp** Porta seriale RS-232: Parità in uscita / Registratore
00BE	FSBLK	190	**Cassette Read / Write Block Count** Contatore dei blocchi di Lettura / Scrittura su registratore

I tendini del Sinclair

La versione Spectrum del programma per il controllo del braccio robot

Il software per il controllo del braccio robot deve necessariamente contenere una sezione in codice macchina. Infatti ai servomotori digitali impiegati occorrono impulsi regolari, generati ogni cinquantesimo di secondo e una simile velocità non è controllabile dai programmi scritti in BASIC. Come nelle versioni per il Commodore e per il BBC, il codice macchina ricorre all'impiego di interrupt per la generazione dei segnali, ma il listato è alquanto diverso essendo impiegato il chip Z80 anziché un processore della serie 65XX. Le interruzioni nel codice macchina dello Z80 sono meglio gestite nel modo di interrupt IM2. In termini semplici, quando viene fissato questo modo operativo e viene generata un'interruzione per aggiornare lo schermo (ciò avviene giusto una volta ogni cinquantesimo di secondo), il processore compone l'indirizzo di una routine di servizio, estraendola da due fonti. Il byte alto dell'indirizzo viene cercato nel registro I del processore, mentre il byte basso viene estratto dal bus dei dati al momento dell'interruzione.

Questo sistema è stato progettato per la comunicazione con periferiche collegate al bus dei dati, in grado di controllare le informazioni immesse su tale bus. In questa applicazione, però, lo stato del bus dei dati è ignoto, così vanno previste tutte le 256 combinazioni. Il programma, pertanto, predispone un'intera pagina di memoria affinché contenga l'indirizzo effettivo della routine di servizio. La pagina, in effetti, viene riempita con il valore #FB. Poiché nel registro I viene caricato il valore #FB, quando si verifica l'interrupt nel modo IM2, l'indirizzo risultante è #FBFB ed è in tale locazione che viene codificata la routine di servizio.

Nel resto del codice macchina vengono adottati gli stessi principi visti già nelle versioni per gli altri computer. Il flusso di impulsi viene emesso dalla porta di I/O 31, qui assegnata all'interfaccia per il braccio. Anche la sezione in BASIC del programma è sulla falsariga delle versioni precedenti, permettendo di comporre sequenze di movimenti agendo tramite la tastiera.

VERSIONE SPECTRUM DEL PROGRAMMA DI CONTROLLO PER IL BRACCIO ROBOT

Caricatore in BASIC

```
10 REM ********************************
15 REM ********************************
17 REM **                              **
20 REM **  Controllo del braccio       **
25 REM **       per Spectrum           **
30 REM **                              **
40 REM ********************************
50 REM ********************************
60 :
70 CLEAR 30000
80 LET sa=64017: REM inizio routine di cuneo
82 LET np=64008: REM indirizzo iniziale di newpos
84 LET dl=64016: REM indirizzo del fattore di ritardo
86 LET en=64148: REM disinserimento del cuneo
88 LET sm=64151: REM indirizzo routine
   movimento scorrevole
90 LOAD "ARM.HEX" CODE sa
100 GO SUB 1000: REM preparazione
110 CLS
120 PRINT AT 6,2;"Opzioni:"
130 PRINT AT 9,2;"1..definiz. nuova sequenza"
140 PRINT AT 10,2;"2..aggiunte alla sequenza"
150 PRINT AT 11,2;"3..load di un file"
160 PRINT AT 12,2;"4..fine lavoro"
170 LET g$=INKEY$: IF g$="" THEN GO TO 170
180 IF g$="1" THEN LET c=1: LET lm=0
190 IF g$="1" OR g$="2" THEN GO SUB 2000: GO SUB
   3000: GO SUB 4000: GO SUB 5000
200 IF g$="3" THEN GO SUB 6000
210 IF g$="4" THEN CLS: RANDOMIZE USR en: STOP
220 GO TO 110
230 :
1000 REM ***** preparazione *****
1010 LET c=1: LET ns=4: REM n. di servomotori
1020 LET lm=0: LET oc=0: LET df=10: REM fattore
   di ritardo
1030 LET mc=100: REM conteggio massimo
1040 DIM r(mc,ns): REM matrice posizione dei tasti
1050 DIM l(1): REM matrice ultimi valori massimi
1060 FOR i=0 TO 3: POKE np+1,0: NEXT i
1070 RANDOMIZE USR sa
1090 RETURN
1099 :
2000 REM ***** informaz. *****
2010 CLS
2020 PRINT AT 3,2;"usare:"
2030 PRINT AT 4,2;"n/m...per s/d"
2040 PRINT AT 5,2;"p/l...s/g del 1mo braccio"
2050 PRINT AT 6,2;"a/z...s/g del 2do braccio"
2060 PRINT AT 7,2;"x/c...azionam. della morsa"
2070 PRINT AT 8,2;"s.....memoriz. di posizione"
2080 PRINT AT 9,2;"q.....ritorno al menu"
2090 PRINT AT 10,2;"r....movim. a posiz. salvata"
2100 PRINT AT 11,2;"v/b..passo avanti/indietro"
2110 PRINT AT 12,2;"e....fissa nuovo passo"
2120 PRINT AT 13,2;"i/d..inc/dec velocita'"
2130 PRINT AT 1,2;"passo=";c;" ";
2135 FOR i=1 TO 4: PRINT r(c,i);" ";:NEXT i: PRINT
2140 RETURN
2190 :
3000 REM ***** programmaz. del braccio *****
3010 POKE dl,1
3020 LET dx=5
3030 LET a$=INKEY$: IF a$="" THEN GO TO 3030
3035 IF a$="n" THEN LET p=PEEK (np)+dx: IF
   p<256 THEN POKE np,p
3040 IF a$="m" THEN LET p=PEEK (np)-dx: IF
   p>0 THEN POKE np,p
3050 IF a$="p" THEN LET p=PEEK (np+1)+dx: IF
   p<256 THEN POKE np+1,p
3060 IF a$="l" THEN LET p=PEEK (np+1)-dx: IF
   p>0 THEN POKE np+1,p
3065 IF a$="a" THEN LET p=PEEK (np+2)+dx: IF
   p<256 THEN POKE np+2,p
3070 IF a$="z" THEN LET p=PEEK (np+2)-dx: IF
   p>0 THEN POKE np+2,p
3080 IF a$="x" THEN LET p=PEEK (np+3)+3*dx: IF
   p<256 THEN POKE np+3,p
3090 IF a$="c" THEN LET p=PEEK (np+3)-3*dx: IF
   p>0 THEN POKE np+3,p
3100 IF a$="r" THEN FOR i=1 TO 4: POKE np+i-1,r
   (c,i): NEXT i
3110 IF a$="v" THEN LET c=c+1: IF c>mc THEN
   LET c=1
3120 IF a$="b" THEN LET c=c-1: IF c<1 THEN
   LET c=mc
3130 IF a$="e" THEN PRINT AT 2,20;"valore del
   passo": INPUT c
3140 IF a$="s" THEN FOR i=1 TO 4: LET r(c,i)=PEEK
   (np+i-1): NEXT i: LET c=c+1
3145 IF a$="i" THEN LET dx=dx+1
3147 IF a$="d" THEN LET dx=dx-1: IF dx<1 THEN
   LET dx=1
3150 IF c>lm THEN LET lm=c
3160 IF oc<>c AND c>1 THEN PRINT AT 1,2;"passo=";
   c-1;" ";: FOR i=1 TO 4: PRINT r(c-1,i);" ";:
   NEXT i: PRINT
3170 LET oc=c
3190 IF a$<>"q" THEN GO TO 3030
3200 RETURN
3900 :
4000 REM ***** ripetizione sequenza *****
4010 CLS
4020 PRINT AT 12,2;"ripeto sequenza s/n,(r ripete)"
4030 LET a$=INKEY$: IF a$<>"s" AND a$ <>"n" AND a$
   <>"r" THEN GO TO 4030
```

```
4040 IF a$="n" THEN RETURN
4050 IF a$="r" THEN PRINT AT 14,1;"fattore di ritardo
     1-255";: INPUT df
4060 IF df<1 OR df>255 THEN GO TO 4050
4070 POKE dl,df: REM fissa registro di ritardo
4080 FOR i=1 TO lm
4090 PRINT AT 1,2;"n. nella sequenza=";i;" ";
4100 FOR s=1 TO 4
4110 POKE np+s-1,r(i,s)
4120 NEXT s: NEXT i
4140 GO TO 4010
4990 :
5000 REM ***** save del file *****
5010 CLS
5020 PRINT AT 12,2;"save del file (s/n)?"
5030 LET g$=INKEY$: IF g$<>"s" AND g$<>"N" THEN GO
     TO 5030
5040 IF g$="n" THEN RETURN
5050 LET l(1)=lm
5060 INPUT "nome file";f$
5065 PRINT "premi play e record"
5067 RANDOMIZE USR en: REM disinserisce cuneo
5070 SAVE f$+".lm" DATA l()
5080 SAVE f$+"r" DATA r()
5085 RANDOMIZE USR sa: REM inserisce cuneo
5090 RETURN
5900 :
6000 REM ***** load di un file *****
6010 CLS
6020 PRINT AT 12,2; "load di un file (s/n)"
6030 LET g$=INKEY$: IF g$<>"s" AND g$<>"n"
     THEN GO TO 6030
6040 IF g$="n" THEN RETURN
6045 INPUT "nome file";f$
6050 FOR i=1 TO mc
6060 FOR j=1 TO ns
6070 LET r(i,j)=0
6080 NEXT j: NEXT i
6090 RANDOMIZE USR en: REM disinserisce cuneo
6100 LOAD f$+".lm" DATA l()
6110 LET lm=l(1): LET c=lm: LET oc=0
6120 LOAD f$+".r" DATA r()
6125 RANDOMIZE USR sa: REM inserisce cuneo
6130 RETURN
```

Listato in Assembly

```
                1000 ;CONTROLLO BRACCIO PER SPECTRUM
                1010
001F            1020 PORT:   EQU  31
F900            1030         ORG  #F900
F900            1040 MOTTAB  DEFS 256
FA00            1050 ANGTAB  DEFS 8
FA08            1060 NEWPOS  DEFS 8
FA10            1070 RITAR:  DEFS 1
                1080
                1090 ;PREPARAZ. TABELLA VETTORI
FA11 2100FC     1100 INIT:   LD   HL,#FC00
FA14 01FB00     1110         LD   BC,#00FB
FA17 71         1120 CICL1:  LD   (HL),C
FA18 23         1130         INC  HL
FA19 10FC       1140         DJNZ CICL1
FA1B 71         1150         LD   (HL),C
FA1C 3EFC       1170         LD   A,#FC
FA1E ED47       1180         LD   I,A
                1190 ;INIZIALIZZA TABELLE CON FF
                1200 ;
FA20 2100F9     1210         LD   HL,MOTTAB
FA23 3EFF       1220         LD   A,#FF
FA25 06FF       1230         LD   B,#FF
FA27 77         1240 CICL2:  LD   (HL),A
FA28 2C         1250         INC   L
FA29 10FC       1260         DJNZ CICL2
FA2B 2100FA     1270         LD   HL,ANGTAB
FA2E 0610       1280         LD   B,16
FA30 77         1290 CICL2A  LD   (HL),A
FA31 2C         1300         INC  L
FA32 10FC       1305         DJNZ CICL2A
FA34 ED5E       1310         IM   2
FA36 C9         1320         RET
                1380 ;TRATTAMENTO INTERRUPT
                1390
FA37 F3         1410 TRATT   DI
FA38 F5         1420         PUSH AF
FA39 C5         1430         PUSH BC
FA3A D5         1440         PUSH DE
FA3B E5         1450         PUSH HL
FA3C FF         1460         RST  #38  ;ROUTINE NORM.
FA3D F3         1470         DI
FA3E CD47FA     1480         CALL EVENTO ;NOSTRA ROUTINE
FA41 E1         1490         POP  HL
FA42 D1         1500         POP  DE
FA43 C1         1510         POP  BC
FA44 F1         1520         POP  AF
FA45 FB         1530         EI
FA46 C9         1540         RET
                1550
                1560 ;
                1570 ;++++ ROUTINE EVENTO ++++
                1580 ;
FA47 0608       1590 EVENTO: LD   B,8
FA49 3E7F       1600         LD   A,#7F
FA4B 16F0       1610         LD   D,#F0
FA4D 2158FA     1620         LD   HL,FIX+2
FA50 DD2100FA   1630         LD   IX,ANGTAB
FA54 3607       1635         LD   (HL),7
FA56 DD5E07     1640 FIX:    LD   E,(IX+7)
FA59 12         1650         LD   (DE),A
FA5A 0F         1660         RRCA
FA5B 35         1670         DEC  (HL)
FA5C 10F8       1680         DJNZ FIX
                1690 ;
                1740 ;PREP. MOTTAB PER INVIO SU PORTA
                1750 ;
FA5E 0600       1760         LD   B,0
FA60 3EFF       1770         LD   A,#FF
FA62 2100F9     1780         LD   HL,MOTTAB
FA65 A6         1790 CICL3:  AND  (HL)
FA66 77         1800         LD   (HL),A
FA67 2C         1810         INC  L
FA68 10FB       1820         DJNZ CICL3
                1830 ;
                1840 ;++++ INIZIO IMPULSI ++++
                1850 ;
FA6A 3EFF       1860         LD   A,#FF
FA6C D31F       1870         OUT  (PORT),A
                1880 ;
                1885 ;++++ CALL RITARDO ++++
FA6E D5         1887         PUSH DE
FA6F 1E04       1889         LD   E,4
FA71 16FF       1890         LD   D,#FF
FA73 CD8BFA     1892         CALL OCICL
FA76 D1         1894         POP  DE
                1920 ;
                1930 ;++++ INVIO MOTTAB ALLA PORTA ++++
                1940 ;
FA77 0E1F       1950         LD   C,PORT
FA79 06FF       1960         LD   B,#FF
FA7B 2E00       1970         LD   L,00
FA7D EDB3       1980         OTIR
                1990 ;
                2000 ;++++ RIPRISTINA MOTTAB A FF ++++
FA7F 3EFF       2010         LD   A,#FF
FA81 0600       2020         LD   B,0
FA83 2100F9     2030         LD   HL,MOTTAB
FA86 77         2040 CICL4:  LD   (HL),A
FA87 2C         2050         INC  L
FA88 10FC       2060         DJNZ CICL4
FA8A C9         2070         RET
                2080 ;++++ CICLO DI RITARDO ++++
                2090 ;
FA8B 1D         2100 OCICL:  DEC  E
FA8C C8         2110         RET  Z
FA8D 15         2120 ICICL:  DEC  D
FA8E CA8BFA     2130         JP   Z,OCICL
FA91 C38DFA     2140         JP   ICICL
                2200 ;
                2210 ;++++ RIPRISTINA IM 1 ++++
                2220
FA94 ED56       2230 REST:   IM   1
FA96 C9         2240         RET
                2250 ;
                2340 ;++++ MOVIMENTO SCORREVOLE ++++
                2350 ;
FA97 0E04       2360 PARTE:  LD   C,4
FA99 0604       2370         LD   B,4   ;PREPARA CONTATORI
FA9B 2103FA     2380         LD   HL,ANGTAB+3
FA9E 110BFA     2390         LD   DE,NEWPOS+3
FAA1 7E         2400 PROX    LD   A,(HL)
FAA2 EB         2410         EX   DE,HL
FAA3 BE         2420         CP   (HL)
FAA4 EB         2430         EX   DE,HL
FAA5 280A       2440         JR   Z,MOSSO ;UGUALE
FAA7 3804       2450         JR   C,SOMM  ;PRESTITO
FAA9 35         2460         DEC  (HL)
FAAA C3B2FA     2470         JP   NEXT
FAAD 34         2480 SOMM:   INC  (HL)
FAAE C3B2FA     2490         JP   NEXT
FAB1 0D         2500 MOSSO:  DEC  C
FAB2 2B         2510 NEXT:   DEC  HL
FAB3 1B         2520         DEC  DE
FAB4 10EB       2530         DJNZ PROX
FAB6 F5         2540         PUSH AF
FAB7 D5         2550         PUSH DE
FAB8 ED5B10FA   2560         LD   DE,(RITAR)
FABC 16FF       2570         LD   D,#FF
FABE CD8BFA     2580         CALL OCICL  ;CALL RITARDO
FAC1 D1         2590         POP  DE
FAC2 F1         2600         POP  AF
FAC3 20D2       2610         JR   NZ,PARTE
FAC5 C9         2620         RET
                3000 ;++++ PREPARA INDIRIZZO ++++
                3010 ;++++ DI SALTO A TRATT  ++++
                3020 ;
FBFB            3030         ORG  #FBFB
FBFB F3         3040         DI
FBFC C337FA     3050         JP   TRATT

Pass 2 errors: 00

SOMM    FAAD    ANGTAB  FA00
RITAR   FA10    EVENTO  FA47
FIX     FA56    TRATT   FA37
ICICL   FA8D    INIT    FA11
CICL1   FA17    CICL2   FA27
CICL2A  FA30    CICL3   FA65
CICL4   FA86    MOTTAB  F900
MOSSO   FAB1    NEWPOS  FA08
PROX    FAA1    NEXT    FAB2
OCICL   FA8B    PORT    001F
REST    FA94    PARTE   FA97
```

Schermi multipli

Testi e grafica in alta risoluzione simultaneamente sullo schermo

Per visualizzare caratteri e simboli, il chip VIC-II deve leggerli dalla memoria e ciò avviene facendogli accedere a parte alle linee del bus degli indirizzi e a tutte le linee del bus dei dati. La procedura di lettura della RAM e della ROM da parte del VIC-II in combinazione con il processore 6510 è chiamata "accesso diretto alla memoria", o DMA (Direct Memory Access). Idealmente, il VIC-II usa le linee degli indirizzi e dei dati nei momenti in cui queste non sono utilizzate dal processore, così le operazioni dei due chip avvengono in modo reciprocamente "trasparente", ossia non interferiscono tra di loro. Sfortunatamente, però, il 6510 è un processore abbastanza semplice: è dotato di pochi registri interni, ma nessuna istruzione impiega più di sette cicli macchina (in media tre o quattro cicli).

Poiché il VIC-II deve leggere grandi quantità di dati, specie quando si tratta di visualizzare più sprite, non c'è tempo per operare in modo trasparente, così il chip è stato dotato di una speciale linea, chiamata BA ("Bus Available", ossia "bus disponibile"), per segnalare al 6510 che il bus è al momento temporaneamente occupato. Solo così il VIC-II può espletare le proprie funzioni di visualizzazione.

Quando la linea BA è posta a livello logico basso il 6510 viene informato che il bus è richiesto dal VIC-II. Al processore viene dato tempo per completare l'istruzione in corso, prima che venga emesso un altro segnale che abilita l'accesso del 6510 ai bus. Questo segnale è chiamato "Address Enable Control" (controllo di abilitazione del bus) o AEC.

La quantità di tempo "rubata" al 6510 è piuttosto grande, considerate le unità di misura in gioco: si pensi che gli indirizzi dei puntatori ai caratteri devono essere letti ogni volta che sullo schermo viene effettuata la scansione di otto linee (i caratteri sono formati da otto righe di pixel) e che ogni linea implica 40 accessi consecutivi per estrarre gli indirizzi dei puntatori alla memoria video (ci sono 40 caratteri su ogni riga di schermo). Nel sistema PAL il chip del video aggiorna ciascuna delle 625 linee dello schermo (procedendo a linee alterne) circa 25 volte al secondo. Il sistema americano NTSC prevede solo 524 linee, ma con una frequenza di aggiornamento di 30 volte al secondo. Tutto ciò richiede del tempo e, con un calcolo approssimato, si può affermare che il 6510 viene rallentato di un buon 15 o 20 per cento.

Queste momentanee "assenze" del 6510 non comportano alcun inconveniente, eccetto quando si debba operare in tempo reale. È il caso, ad esempio, delle operazioni di lettura o scrittura su nastro, in cui la sincronizzazione è fondamentale. Ecco perché si rende necessario bloccare la visualizzazione su schermo (con POKE 53265,11) prima dell'operazione di ingresso/uscita, per riabilitarla in seguito (con POKE 53265,27). La necessità di sospendere la visualizzazione dipende dal modo di funzionare della periferica con la quale il computer intende comunicare: nelle operazioni sui dischetti, ad esempio, non occorre disabilitare lo schermo.

Suddividere lo schermo

Sul Commodore 64, con qualche accorgimento di programmazione, è possibile suddividere lo schermo in modo da adoperare contemporaneamente la grafica sia in alta sia in bassa risoluzione. Come accade spesso nei giochi in commercio, la metà superiore dello schermo raffigura uno scenario in grafica ad alta risoluzione, mentre in quella inferiore sono visualizzati i messaggi e l'input del giocatore. Normalmente non è possibile mescolare i due tipi di grafica.

Il programma *Splitscreen* presentato più avanti, che illustra molti dei concetti fin qui esposti, opera in modo abbastanza semplice: nella RAM "nascosta" sotto la ROM del BASIC viene creato uno schermo in alta risoluzione, che viene ininterrottamente visualizzato nel terzo superiore dello schermo. Al tempo stesso, i due terzi inferiori rimangono nella normale grafica a bassa risoluzione. A pag. 1618 si è visto come usare i vettori RAM sul Commodore 64, modificandone l'indirizzo affinché vengano eseguite routine in codice macchina scritte dall'utente. Per suddividere lo schermo qui viene operato un intervento, piuttosto insolito, sul vettore IRQ. La routine abbinata a questo vettore viene chiamata in diverse occasioni, una delle quali si presenta quando un temporizzatore di uno dei CIA modifica un bit nel registro segnalatore delle interrupt (interrupt flag register, o IFR), che ha per indirizzo 53273 ($D019). Ciò avviene circa ogni sessantesimo di secondo, provocando l'esecuzione della routine di servizio IRQ. Tra le altre cose, questa routine effettua una scansione della tastiera, per vedere se si sta premendo un tasto. Alcuni bit del registro IFR vengono posti a uno anche a seguito di certi eventi controllati dal chip VIC-II, per esempio quando avviene una collisione tra due sprite o quando il contatore di scansione raggiunge un valore prefissato. Esiste un altro registro che influenza la linea IRQ del 6510: il registro di abilitazione delle interrupt (interrupt enable register, o IER), il cui indirizzo è 53274 ($D01A). Se il programmatore modifica i bit di IER, allora i corrispondenti bit di IFR provocano una chiamata alla routine di IRQ.

Ecco i principi del programma *Splitscreen*:

1) Quando si verifica l'interruzione nella scansione in cima allo schermo, il programma seleziona il modo "mappa di bit", predispone un'interruzione di scansione a

La prima interruzione di scansione provoca il passaggio all'alta risoluzione



La seconda interruzione di scansione causa il ritorno al modo testuale

DIMENSION GRAPHICS

Una storia in due parti

Moltissimi giochi d'avventura reperibili in commercio per il Commodore 64 si avvalgono di una tecnica basata sulle interruzioni di scansione per visualizzare contemporaneamente la grafica sia in alta sia in bassa risoluzione. L'illustrazione è tratta dal gioco Spiderman (scritto da Scott Adams) e mostra come il terzo superiore dello schermo viene usato per disegnare un ambiente dell'avventura, mentre nella parte restante di schermo compaiono scritte e messaggi nel normale modo testuale. In questo caso vengono sfruttate due interruzioni di scansione: una in cima allo schermo e una al termine della "finestra" in alta risoluzione



circa metà dello schermo e poi esegue un ritorno da interrupt (RTI).

2) Quando si verifica l'interruzione di scansione a metà schermo, il programma riseleziona la bassa risoluzione e predispone un'interruzione di scansione all'inizio dello schermo. Di nuovo esegue un ritorno da interrupt (RTI).

Si noti la curiosa configurazione di qujfo sistema. varte della normale memoria video (corrispondente al terzo superiore dello schermo) viene utilizzata per conservare informazioni di colore per la grafica in alta risoluzione, mentre il resto dello schermo è adoperato normalmente in bassa risoluzione. Rimane però un inconveniente da superare. Le interruzioni di scansione si verificano circa ogni cinquantesimo di secondo e devono essere "servite" immediatamente, altrimenti nel frattempo il raggio di scansione prosegue nel suo spostamento e i dati verrebbero visualizzati in posizione errata. Normalmente la routine di IRQ viene innescata ogni sessantesimo di secondo e la sua esecuzione ha una durata relativamente lunga. Se questa inizia subito prima di un'interruzione di scansione, c'è il rischio che la routine di "cuneo", inserita dal nostro programma, venga eseguita con troppo ritardo, provocando un fastidioso sfarfallìo sul margine di separazione tra i due schermi.

La soluzione è abbastanza semplice: si tratta di forzare l'esecuzione della routine di IRQ subito dopo quella della routine di cuneo relativa alla scansione. Ciò si ottiene spegnendo il normale temporizzatore per i sessantesimi di secondo (disabilitando così il normale innesco della routine IRQ) e saltando alla routine IRQ subito dopo aver eseguito il "cuneo". Con questo accorgimento si garantisce la necessaria sincronizzazione, anche se ciò implica una scansione leggermente meno frequente della tastiera che, del resto, non ha alcuna conseguenza pratica.

Infine, rimane da collocare lo schermo in alta risoluzione nella RAM normalmente "nascosta" dalla ROM del BASIC. L'unico modo per accedere convenientemente a questa RAM, adoperando il BASIC, è ricorrere a delle POKE, ma se il programma è in codice macchina l'intera ROM del BASIC non serve e può essere disabilitata liberando così 8 Kbyte di memoria. Comunque, con qualche modifica al listato qui riportato, è possibile spostare altrove lo schermo in alta risoluzione, per potervi accedere anche in ambiente BASIC.

PROGRAMMA SPLITSCREEN PER COMMODORE 64

Caricatore in BASIC

```
900 REM*******************************
902 REM**   CARICATORE BASIC PER    **
903 REM**      SPLITSCREEN 64       **
910 REM** NOTA: IN QUESTA VERSIONE **
920 REM** LA HIRES E' SOTTO BASIC  **
930 REM** E NON SI PUO' USARE PEEK **
940 REM** NELLA SUBROUTINE DI PLOT **
950 REM*******************************
960 :
1000 REM CARICAMENTO SPLIT SCREEN E PROVA **
1010 DATA76,50,192,49,234,173,25,208
1020 DATA240,37,169,255,160,21,141,25
1030 DATA208,173,0,221,73,2,141,0,221
1040 DATA173,17,208,73,32,141,17,208,41
1050 DATA32,240,4,169,121,160,255,141
1060 DATA18,208,140,24,208,108,3,192
1070 DATA173,14,220,41,254,141,14,220
1080 DATA173,20,3,141,3,192,173,21,3
1090 DATA141,4,192,169,5,141,20,3,169
1100 DATA192,141,21,3,173,17,208,41,95
1110 DATA141,17,208,169,0,141,18,208
1120 DATA169,255,141,25,208,169,1,141
1130 DATA26,208,165,1,41,254,133,1,169
1140 DATA0,133,247,169,160,133,248,160
1150 DATA0,162,28,169,0,145,247,200,208
1160 DATA251,230,248,202,208,246,169
1170 DATA188,133,248,160,0,162,4,169
1180 DATA183,145,247,200,208,251,230
1190 DATA248,202,208,246,165,1,9,1,133
1200 DATA1,96,255
1210 DATA20633:REM*CHECKSUM*
1220 CC=0:FOR I=0 TO 160
1230 READ X: POKE 49152+I,X: REM INSERIM.
     CODICE
1240 CC=CC+X: NEXT
1250 READ X: IF X<>CC THEN PRINT "ERRORE
     CHECKSUM": STOP
1260 REM*** VERIFICA SUDDIVISIONE SCHERMO ***
1270 SYS49152: REM CHIAMATA A SPLIT SCREEN
1280 CM = 40960: REM INIZIO HIRES
1290 REM** TRACCIA ASSI VERT. **
1300 X=160: FOR Y=0 TO 70
1310 GOSUB 1400: REM CALCOLA INDIRIZZO
1320 POKE M+R,2^(7-C): REM TRACCIA PUNTO
1330 NEXT
1340 REM** TRACCIA ASSI ORIZZ. **
1350 Y=37: FOR X=0 TO 319
1360 GOSUB 1400: REM CALCOLA INDIRIZZO
1370 POKE M+R,255: REM TRACCIA SEGMENTO
1380 NEXT
1390 END
1400 REM** S/R DI CALCOLO **
1410 U=INT(Y/8): V=INT(X/8)
1420 R=Y AND 7: C=X AND 7
1430 M=CM +' (40*U+V)*8
1440 RETURN
```

Listato in Assembly

```
;++++++++++++++++++++++++++
;++++++++++++++++++++++++++
;++                      ++
;++    SPLITSCREEN 64    ++
;++                      ++
;++++++++++++++++++++++++++
;++++++++++++++++++++++++++

      *= $C000
      JMP INIZIO
      MEM1 = $F7
      MEM2 = $F9
      MEM3 = $FB
      SCN  = $FD
      CINV = $314     ;VETTORE IRQ
      TEMPIRQ *=*+2
NEWIRQ **             ;NUOVO INIZIO CUNEO IRQ
      LDA $D019       ;ESAMINA STATO INTERRUPT      DDD
      BEQ NOTVIC      ;IRQ PROVIENE DAL VIC?
      LDA #$FF
      LDY #$15
      STA $D019       ;AZZERA MEMORIA INTERRUPT
      LDA $DD00
      EOR #$02
      STA $DD00       ;SCAMBIO DEI BANCHI DI MEM.
      LDA $D011
      EOR #$20        ;CAMBIA IL BIT DEL BIT MAP MODE
      STA $D011
      AND #$20        ;ESAMINA IL BIT DEL BMM
      BEQ LSCN        ;PARTE INFERIORE DELLO SCHERMO  DDE
      LDA #$79        ;FISSA SCANSIONE ALLA LINEA 22
      LDY #$FF
LSCN
      STA $D012       ;DEPOSITA N. VAL. DI SCANSIONE
      STY $D018
NOTVIC
      JMP (TEMPIRQ)
INIZIO **             ;INIZIALIZZA IL CUNEO IRQ
      AND #$FE        ;DISABILITA IL TIMER A
      STA $DC0E       ;CIA #1
      LDA CINV
      STA TEMPIRQ     ;
      LDA CINV+1      ;CAMBIA IL VETTORE IRQ
      STA TEMPIRQ+1   ;
      LDA #<NEWIRQ
      STA CINV
      LDA #>NEWIRQ
      STA CINV+1
      LDA $D011
      AND #$5F        ;RESET DEL BIT MODE
      STA $D011
      LDA #$00
      STA $D012
      LDA #$FF
      STA $D019       ;RESET MEMORIA IRQ
      LDA #$01
      STA $D01A       ;ABILITA CONFRONTO SCANSIONE
      LDA #01
      AND #$FE
      STA $01         ;RIMUOVE IL BANCO ROM DEL BASIC
      LDA #$00
      STA MEM1        ;PREPARA I PUNTATORI IN PAG. 0
      LDA #$A0        ;PER AREA HIRES "DIETRO" LA ROM
      STA MEM1+1      ;DEL BASIC
      LDY #$00
      LDX #$1C
      LDA #$00        ; HIRES

      STA (MEM1),Y
      INY
      BNE DDD
      INC MEM1+1
      DEX
      BNE DDD
      LDA #$BC
      STA MEM1+1
      LDY #$00
      LDX #$04
      LDA #$B7        ;FISSA COLORE

      STA (MEM1),Y
      INY
      BNE DDE
      INC MEM1+1
      DEX
      BNE DDE
      LDA $01
      ORA #$01
      STA $01         ;RIPRISTINA ROM DEL BASIC
      RTS             ;RITORNO AL BASIC
```

Impossible Mission

Un gioco con una grafica eccellente, dotato di sintesi vocale

Moltissimi giochi di tipo arcade prevedono movimenti di passaggio da una piattaforma ad un'altra sullo schermo. Il giocatore, facendo balzare qua e là la figurina protagonista deve farle toccare o raccogliere oggetti vari. In genere il protagonista fa il suo ingresso da un punto situato in basso e deve raggiungere i livelli alti tramite scale, trampolini e via dicendo. Di solito gli oggetti che consentono di guadagnare punti sono situati in posizioni difficili da raggiungere: occorre molta abilità e una certa dose di concentrazione.
Giochi di questo genere sono quasi sempre corredati da "omini" o animali che ostacolano il più possibile l'azione o che mettono in pericolo la "vita" del protagonista. Questi avversari possono essere mobili o statici e in grado di lanciare oggetti oppure no. A seconda delle intenzioni di chi ha realizzato il programma i movimenti degli avversari possono essere più o meno intelligenti. In certi casi gli spostamenti sono solo orizzontali o verticali, in altri casi tendono a incapsulare il protagonista, giungendo fino a precludergli ogni scampo.
Dal punto di vista del programmatore i giochi di piattaforma offrono diversi vantaggi, il principale dei quali è che la codifica necessaria è molto semplice. Infatti, una volta create le immagini delle varie componenti (protagonista, avversari ed elementi di scenario), si tratta semplicemente di collocarli nelle posizioni desiderate. Ciò lascia ampio spazio per migliorare i dettagli del gioco o per aumentare il numero di ambienti. Un ulteriore risparmio di memoria si ottiene riadoperando immagini modulari per comporre più figure complesse.
Il gioco *Impossible Mission* si avvale di tutte queste tecniche, tipiche dei giochi di piattaforma.
Scopo del gioco è quello di recuperare i vari frammenti di un codice (nascosto nell'arredamento di ben 32 stanze), che consente al giocatore di entrare nel nascondiglio di un malvagio scienziato, chiamato Elvin.
A guardia degli elementi di arredamento ci sono bellicosi robot e, occasionalmente, compare una grossa sfera nera. Nell'arredamento sono nascoste anche parole segrete che, individuate, permettono al giocatore di disattivare temporaneamente i robot e di azionare gli ascensori tra le varie stanze. Ogni parola segreta va inserita nel terminale di computer presente in ogni stanza del gioco. Nella parte inferiore sinistra dello schermo è riprodotta una piantina della casa, in cui è segnalata la posizione corrente del giocatore.
Le parole segrete si possono anche ottenere dalle "stanze del codice": la stanza emette una serie di note che il giocatore deve disporre in ordine ascendente. Più parole segrete si intendono carpire, più sono le note emesse. Diversamente da altri giochi di piattaforma, il giocatore non ha un numero limitato di "vite", bensì ad ogni "decesso" diminuisce il tempo di gioco.
Nel codificare questo gioco i programmatori hanno saputo sfruttare egregiamente la memoria, tanto che sono riusciti ad inserire anche un commento parlato ottenuto con sintesi vocale. Un fatto quasi incredibile se si considera che, in genere, per memorizzare appena un paio di frasi di una decina di parole occorrono dai tre ai quattro Kbyte di RAM. Per di più la qualità del parlato (in inglese, ovviamente) è considerevolmente superiore che in molti altri giochi. Anche la grafica è molto buona, nonostante quella degli ambienti sia leggermente trascurata.

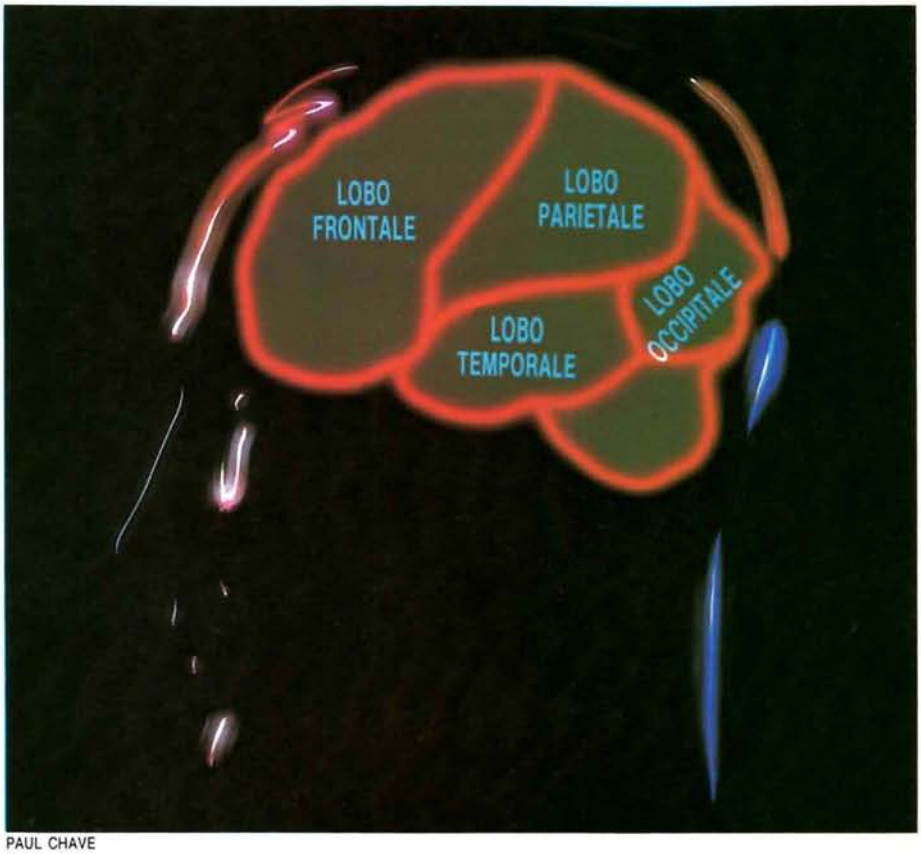


Niente è impossibile!
Scopo del gioco è quello di raggiungere il nascondiglio di Elvin, uno scienziato malvagio, attraversando una casa composta da 32 stanze. Entrati in una stanza si devono cercare i frammenti del codice segreto che consente l'accesso al nascondiglio. Nel far ciò vanno tenuti a bada bellicosi robot posti a guardia dei vari ambienti

Impossible Mission: Per il Commodore 64
Produttore: CBS Software
Autore: Dennis Caswell
Joystick: Necessario
Formato: Cassetta o disco

Intelligenza e cervello

Teoria e pratica dell'Intelligenza Artificiale con i microcomputer



PAUL CHAVE

Agli inizi degli anni '70 l'Intelligenza Artificiale (spesso abbreviata in AI, dalla dizione inglese) era considerata la "Cenerentola" dell'Informatica e giudicata come una branca di scienza adatta ai più lunatici. Di recente, però, sta ottenendo successi enormi richiamando l'attenzione anche dei più scettici. Perfino alcuni prodotti software in commercio si vantano di possedere procedure di AI.

Per comprendere la situazione attuale di questa scienza (e prevedere quella futura), occorre naturalmente rifarsi alle sue origini. La storia dell'Intelligenza Artificiale può essere suddivisa all'incirca in quattro periodi ciascuno dei quali, della durata di circa 10 anni, distinto per un particolare tema predominante. Questa suddivisione implica necessariamente una semplificazione, ma permette di evidenziare i fatti più importanti. Ognuno dei quattro temi può essere considerato come risposta alla domanda: «Di che cosa si occupa l'Intelligenza Artificiale?», posta ai ricercatori di ciascun periodo:

Anni '50: Le reti neurali
Anni '60: La ricerca euristica
Anni '70: I sistemi esperti
Anni '80: L'apprendimento da parte di macchine

Nel 1943, Warren McCulloch e Walter Pitts proposero un modello per il neurone del cervello umano e animale. Queste cellule nervose, astratte, fornivano la base per una rappresentazione simbolica e matematica dell'attività mentale. Altri ricercatori, tra i quali Norbert Wiener, elaborarono simili intuizioni in un campo che divenne poi noto col termine "cibernetica". Questa scienza è basata sulla convinzione che sia possibile costruire una macchina "senziente", con metodi di analisi e di feedback. Negli anni '50 dalla cibernetica nasce l'Intelligenza Artificiale.

I primi ricercatori di AI adoperarono la teoria di McCulloch come base e, considerata l'enorme complessità del cervello, non c'è da stupirsi se tutti i loro tentativi di generare un sistema intelligente fallirono. L'atteggiamento generale era riassumibile in una frase: "Il cervello è un risolutore intelligente di problemi: simuliamo il funzionamento del cervello". Ovviamente a quei tempi sia l'hardware che il software erano del tutto inadeguati per un progetto così ambizioso.

Uno dei pochi sistemi funzionanti del tempo è stato il Perceptron di Rosenblatt: si trattava di un elementare sistema di visione, al quale si poteva insegnare a riconoscere alcune configurazioni. In effetti il Perceptron consisteva in una fitta serie di cellule fotosensibili, con cui si tentava di simulare il comportamento della retina di un occhio umano. Collegata alle cellule, c'era una serie di elementi "individuatori di configurazione" (chiamati "demoni"), che verificavano lo stato acceso/spento di ciascuna parte dell'immagine. Il segnale, da loro emesso, veniva passato ad un circuito di controllo, in grado di prendere alcune decisioni. Il circuito amplificava e sommava i segnali in uscita dai demoni per produrre un risultato "ponderato": se il risultato superava un valore prefissato, il circuito segnalava una condizione positiva (acceso), altrimenti negativa (spento). Un Perceptron è così in grado di distinguere tra due classi di immagini, benché il principio possa essere esteso ulteriormente.

Le ottimistiche speranze iniziali, che vedevano nel Perceptron la soluzione ad un'infinità di problemi, svanirono ben presto e i ricercatori di AI iniziarono a considerare il pensiero umano come una coordinazione di operazioni su semplici elementi simbolici. Benché si trattasse di un repentino cambiamento di percorso, gli esperti sentivano di muoversi su di un terreno più solido: i computer possono facilmente operare su simboli, ricercandoli e confrontandoli. La parte più difficile era però raggruppare coerentemente un certo numero di attività di semplicità simile.

I ricercatori di maggior rilievo, negli anni '60, sono stati Alan Newell e Herbert Simon dell'Università Carnegie-Mellon. Essi lavorarono, tra l'altro, sulla dimostrazione di teoremi e sul

Menti meccaniche

L'Intelligenza Artificiale si occupa della realizzazione di sistemi computerizzati in grado di svolgere attività altrimenti svolte dall'uomo. Gli scopi di questa scienza, però, sono andati estendendosi e includono anche la percezione visiva e auditiva. In sostanza si tratta sempre di un antico sogno: quello di realizzare una macchina veramente pensante

gioco degli scacchi tramite computer. Il loro principale risultato fu un programma chiamato GPS (da "General Problem Solver", ossia "risolutore generale di problemi"). Il termine "generale" era vero purché venisse definito un preciso "ambiente d'azione", ossia gli elementi che lo compongono e gli operatori da applicare a tali elementi. Comunque, le attività di questo programma si riferivano soprattutto alla soluzione di modesti rompicapo in cui le regole erano relativamente poche e ben definite (un esempio è il gioco delle *Torri di Hanoi*, vedere a pag. 954). Potevano essere risolti anche problemi di carattere matematico-criptoaritmetico, in cui dei micromondi venivano formalizzati tramite una serie di parametri. Il GPS non poteva però risolvere quelli che sono chiamati problemi del mondo reale come diagnosi mediche o previsioni finanziarie basate sulla fluttuazione della moneta.

Il principio alla base del GPS era che un qualsiasi problema si potesse risolvere eseguendo una scansione tra tutte le soluzioni possibili (o accettabili). Per rendere la ricerca più efficiente, il sistema doveva essere pilotato da regole euristiche (ossia essere in grado di apprendere dalla propria esperienza) per trovare il giusto percorso proseguendo per tentativi. Ciò significa che, ipotizzando la totale ignoranza sul percorso di un labirinto, la ricerca della via d'uscita sarebbe durata un periodo di tempo decisamente inaccettabile. Se, invece, alla macchina veniva comunicato qualche elemento indicativo, l'attesa per una soluzione poteva avere tempi più ragionevoli (anche se i processi euristici non garantiscono sempre il successo, poiché talvolta si addentrano in vicoli ciechi). Durante questo periodo, i ricercatori di AI idearono svariate strategie di ricerca euristica, delle quali riparleremo in seguito.

IL General Problem Solver, come si è detto, non era particolarmente adatto a risolvere i problemi del mondo reale, così negli anni '70 un'équipe della Stanford University, guidata da Edward Feigenbaum, pensò di ovviare a questo difetto. Piuttosto che computerizzare gli aspetti generici dell'intelligenza, concentrarono gli sforzi su settori specifici dello scibile: nacquero così i primi "sistemi esperti". Il primo sistema esperto è stato il *Dendral*, un interprete di spettrogrammi realizzato già nel 1967; quello che ha avuto la maggiore influenza, però, è stato il *Mycin*, nel 1974. Il *Mycin* riesce a diagnosticare la presenza di infezioni batteriche in campioni di sangue e a prescrivere la terapia farmacologica adatta. Il *Mycin* ha generato tutta una famiglia di sistemi diagnostici, alcuni dei quali sono oggi regolarmente applicati in ospedali e cliniche specializzate. *Puff*, ad esempio, un sistema diagnostico che riguarda i polmoni, deriva proprio dal *Mycin* ed è usato con profitto dal Pacific Medical Center di San Francisco.

Il *Mycin* ha introdotto una serie di innovazioni che sono poi divenute pietre miliari per quanto concerne i sistemi esperti. In primo luogo la sua "conoscenza" è fondata su centinaia di regole, simili alle seguenti:

REGOLA N. 47:
SE
1) Il luogo della coltura è il sangue e
2) l'identità dell'organismo non è nota con certezza e
3) la colorazione dell'organismo è gram-negativa e
4) la morfologia dell'organismo è "a bastoncino" e
5) il paziente è seriamente ustionato

ALLORA esiste una certa probabilità (0.4) che l'identità dell'organismo sia pseudomonas.

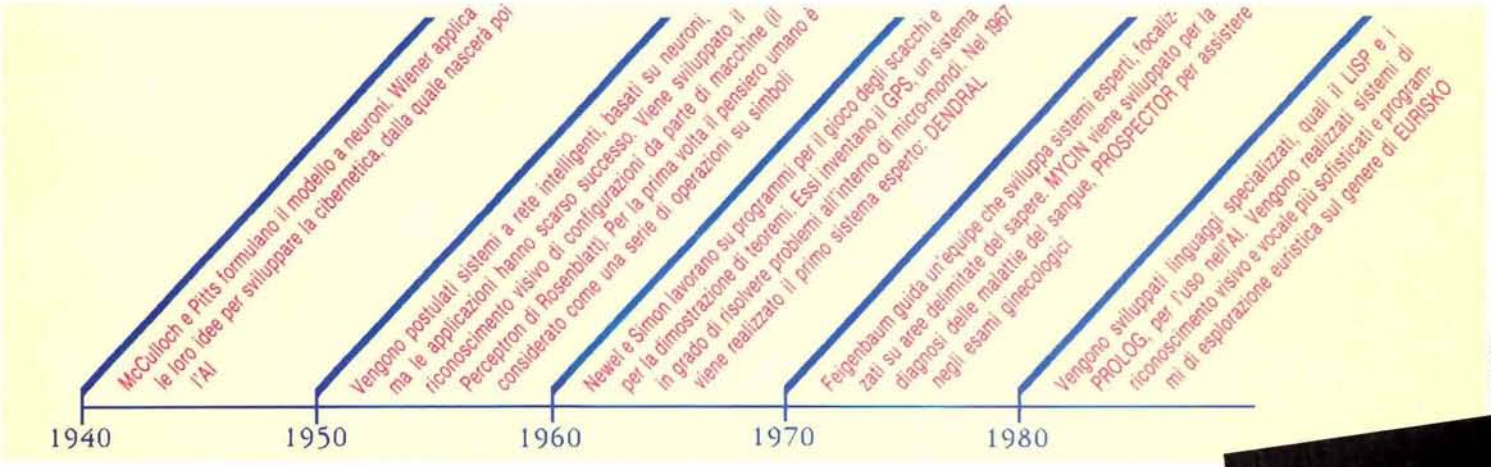
In secondo luogo poi, tali regole sono probabilistiche. Shortliffe, il medico inventore di *Mycin*, ha progettato l'intero schema cercando di ottenere dal sistema conclusioni plausibili, basate su fattori di certezza, partendo da informazioni incerte.

Pertanto il valore 0.4 visto sopra non è propriamente un fattore di probabilità quanto di approssimazione. Il punto più significativo, comunque, è che il *Mycin* e i sistemi simili possono giungere ad una conclusione corretta, anche partendo da informazioni iniziali incomplete o parzialmente errate.

Adoperando metodi di ragionamento "approssimato" assieme ad elementi di calcolo, formulano soluzioni che si avvicinano molto alla ve-

Prospetto cronologico

L'Intelligenza Artificiale è diventata un'area della ricerca scientifica che estende ovunque le sue applicazioni. La storia dell'AI, comunque, è relativamente breve e può essere facilmente riassunta in questo specchietto



LIZ DIXON



Per concessione da COMPUTER WEEKLY

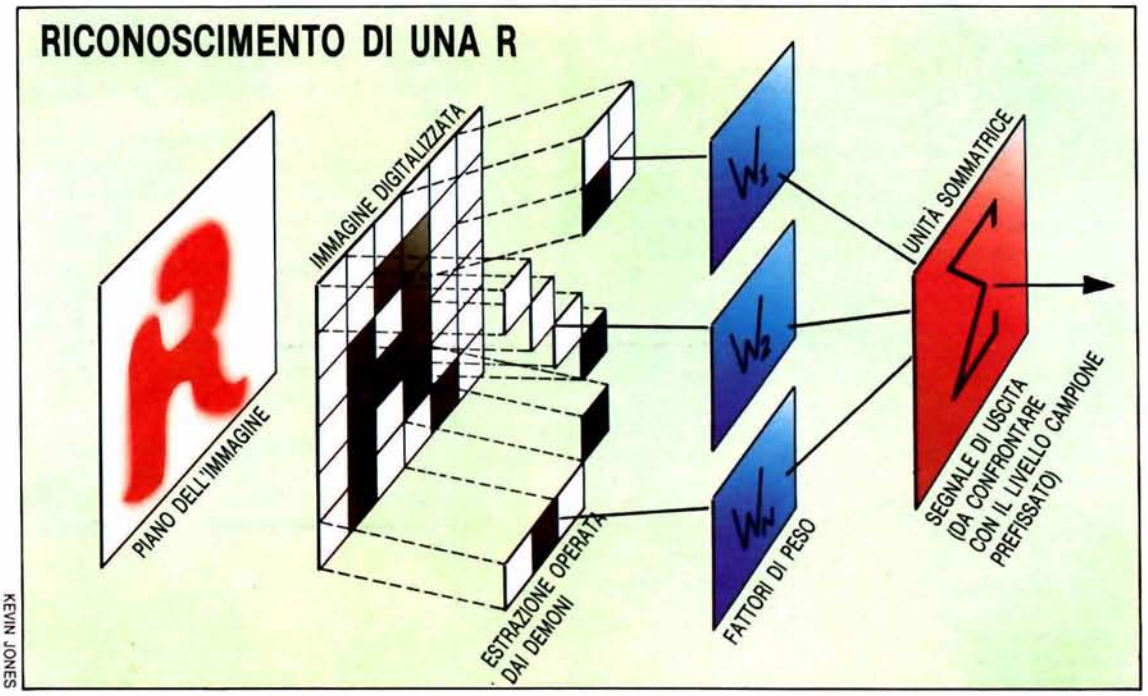
Edward Feigenbaum

Alla guida di un'équipe di ricercatori presso l'Università di Stanford, California, Edward Feigenbaum ha sviluppato i primi sistemi esperti. Questi rappresentarono una grossa innovazione, poiché da allora lo studio si è concentrato sulla creazione di sistemi focalizzati su argomenti ben delimitati, anziché la conoscenza in generale

rità, nonostante l'imperfezione dei dati forniti inizialmente.
In terzo luogo, *Mycin* può fornire una spiegazione del "ragionamento" seguito e l'operatore medico può interrogare il sistema con svariati criteri, anche per chiedere come esso sia giunto ad una data conclusione. Il sistema risponde con una descrizione del processo deduttivo adoperato nel caso specifico. Questo comportamento "amichevole" è un diretto derivato della programmazione impostata su una serie finita di regole.
Infine (fatto molto importante), *Mycin* funziona davvero, macinando rapidamente informazioni che ad un essere umano richiederebbero anni di tirocinio. In realtà il *Mycin* viene usato soprattutto per l'insegnamento, piuttosto che per elaborare diagnosi, ma l'interesse di enti e di governi per l'applicazione reale di simili sistemi va, ovviamente, crescendo. Se poi si prenderà in considerazione la possibilità di forti guadagni in questo settore, l'AI perderà quella "innocenza di intenti" che finora l'ha caratterizzata.

Lo stato dell'arte

Si arriva così agli anni '80. La moda dei sistemi esperti si è già diffusa. L'ingrediente principale è la conoscenza; infatti è lo scopo e la qualità della conoscenza "incorporata" che determina il successo di un sistema esperto.
Tuttavia, la conoscenza non si può spremere come dentifricio da un tubetto: codificare abilità tipicamente umane è un lavoro molto impegnativo e delicato, che richiede parecchio tempo.
Così, mentre tutto il mondo si stupisce della bravura dei sistemi esperti, la scienza ha di nuovo spostato la sua mira: si tratta ora di sviluppare metodi che consentano alle macchine di apprendere, ossia di "apprendimento automatizzato". Come si è già visto, l'AI rappresenta un bersaglio mobile e attualmente, al suo centro, si trova un programma chiamato *Eurisko*.
Eurisko è un programma pressoché autodidatta, che estende e migliora il proprio "bagaglio" di regole automaticamente, per induzione. Oltre a vincere per tre anni di seguito al gioco *Trillion Credit Squadron* (nonostante nel frattempo le regole siano state modificate per ostacolare le vittorie), il programma *Eurisko* è stato applicato anche a problemi più pratici, come l'invenzione di una nuova porta logica tri-dimensionale, nel campo dei circuiti integrati. Indubbiamente i sistemi come *Eurisko* sono l'attuale punta di diamante dell'Intelligenza Artificiale e, poiché la stessa AI è il cavallo di battaglia dell'informatica, questo è il settore verso il quale volgeremo l'attenzione nel prossimo futuro. Per ironia della sorte, dopo tutti questi anni l'AI sembra tornare alle radici: il processo di apprendere, infatti, era considerato il principale problema della cibernetica.
In questa serie dedicata all'AI vengono trattati gli aspetti teorici pratici di questa scienza e sono presentati, più avanti, alcuni programmi che illustrano (in maniera ridotta, considerata la memoria a disposizione) alcuni principi dell'AI sui soliti home computer (Micro BBC, Commodore 64 e Spectrum). Anche se i linguaggi più usati, nell'ambito dell'Intelligenza Artificiale, sono il LISP e il PROLOG, i programmi presentati sono scritti in BASIC, molto più diffuso sugli home computer. Nonostante ciò, essi presentano delle buone approssimazioni della realtà (in fondo un micro di oggi ha una capacità elaborativa pari agli apparecchi di venti anni fa).



KEVIN JONES

Percezioni del Perceptron
L'immagine campione (in questo caso una lettera R) viene proiettata su di un piano e digitalizzata. I cosiddetti "demoni" campionano un ristretto gruppo di pixel (ad esempio 4 alla volta) e reagiscono se la configurazione rilevata corrisponde a una di quelle per le quali sono programmati. Il risultato (0 o 1) fornito da ciascun demone viene poi moltiplicato per un fattore di "peso" (in base all'importanza della sezione di lettera riconosciuta) e la somma di tutti i risultati viene quindi confrontata con un valore campione. Se la somma supera tale valore, il sistema considera come riconosciuta la forma originale, altrimenti no. Sia i fattori di peso che il livello campione possono essere variati durante il processo che serve per insegnare al Perceptron le varie forme di base

Scegliere il giusto DBM

Le domande più frequenti sui programmi per la gestione di archivi

Quali sono i criteri principali che devono guidare alla scelta di un DBM?

Quando si usa un word processor, le operazioni più lunghe e fastidiose sono la ricerca dei file e lo spostamento di blocchi di testo (simile a una procedura di riordinamento). Tuttavia, solo di rado tali operazioni devono essere ripetute, quindi occupano soltanto una piccola porzione del tempo occupato dal lavoro, che viene in gran parte dedicato all'immissione del testo e alla fase di stampa.
Nel caso dei DBM la situazione è diametralmente opposta, poiché i record, in genere, sono "documenti" di piccole dimensioni la cui immissione richiede poco tempo, mentre le operazioni che durano di più sono l'ordinamento e l'estrazione selezionata dei record. Se, come spesso accade, il file dei dati è troppo grande per essere interamente contenuto in RAM, occorrono operazioni, relativamente lunghe, di lettura e scrittura sulla memoria di massa. Queste risultano ancora più lente se viene usato un registratore (memoria di massa sequenziale). Se il database che si intende gestire non è di piccole dimensioni, è consigliabile acquistare un sistema a dischetti.
Molti database vengono impiegati in attività commerciali di vario genere, per le quali vige il detto «il tempo è denaro!». Quindi, se si ha veramente bisogno di un DBM, è necessario un sistema a dischetti (anche se, in certi casi, può bastare un sistema quale il QL della Sinclair con Microdrive).

Quanti record può gestire un DBM? Conviene acquistarne uno con un numero di record maggiore rispetto alle necessità previste?

Se il database deve gestire una notevole quantità di record, conviene scegliere un DBM capace di trattare un numero di record almeno doppio di quello previsto. Se il database viene usato per inventariare e gestire articoli di un magazzino e questi non sono molti, non è necessario acquistare un DBM capace di trattare ben 32.000 record. Tuttavia un "package" di tali dimensioni può non essere sufficiente per la gestione di una biblioteca.
I DBM in grado di mantenere più di 64.000 record sono piuttosto rari e richiedono l'impiego di potenti elaboratori con memorie a disco fisso di grande capacità.

Che tipo di DBM conviene acquistare se non è possibile prevedere in anticipo il numero di campi di cui si ha bisogno

La maggior parte dei DBM pongono un limite massimo al numero dei campi di un record. Tuttavia non sempre è possibile prevedere quanti campi occorrono. Altre volte il numero dei campi consentiti dai DBM è più che sufficiente, anche se la loro lunghezza è limitata ad una sola linea e tale restrizione complica notevolmente l'uso di campi contenenti testi. Una limitazione ancor più comune riguarda il numero di caratteri consentiti per ogni record. Un valore tipico è quello di 1024 caratteri per record: è l'utente che poi decide come dimensionare i vari campi del record e quanti caratteri assegnare a ciascuno di essi. In tutti i casi, si deve ricordare che il numero massimo di caratteri in un campo e di campi in un record deve alla fine consentire la creazione del numero di record desiderati.

Se è vero che i campi chiave hanno una particolare rilevanza, quanti se ne devono creare?

I campi chiave sono dei campi privilegiati, in quanto vengono più facilmente individuati ed elaborati dal DBM. Se a un campo non è stata attribuita la funzione di chiave, non lo si può usare per estrarre i record in base al suo contenuto. I seguenti esempi illustrano qual'è il normale impiego dei campi chiave:

```
SEARCH FORNITORE FOR "Ditta X"
```

Vengono estratti soltanto i record nei quali il campo chiave FORNITORE corrisponde alla stringa di caratteri indicata nell'istruzione.

```
SEARCH PREZZO > = 10.000
```

In questo caso viene esaminato il campo chiave PREZZO di ogni record per individuare i valori maggiori o uguali a 10.000. Alcuni DBM consentono di specificare come "chiavi", durante la definizione del record, un qualsiasi numero di campi (anche tutti), altri invece pongono un limite massimo. L'esperienza insegna che conviene sempre definire il numero massimo di campi chiave consentito.

Alcuni DBM possiedono un proprio linguaggio di programmazione incorporato: è una limitazione o ci sono altre possibilità?

I linguaggi incorporati in alcuni DBM permettono di eseguire automaticamente e in sequenza sofisticate operazioni. In applicazioni di tipo gestionale, questi linguaggi consentono di sviluppare programmi che fanno risparmiare all'operatore una gran mole di lavoro.
Il *dBase II* e l'*Archive* ne sono un eccellente esempio. Se, per particolari esigenze, il database richiede l'esecuzione ripetitiva di sequenze del tipo: «stampa tutte le vendite della giornata seguita da una lista di tutti gli articoli in giacenza e da una seconda lista di tutti gli articoli che risultano al di sotto del livello di riordino», in questo caso conviene acquistare un DBM dotato di un linguaggio interno di programmazione.
Se, d'altra parte, le modalità di accesso al database variano da un giorno all'altro, sono sufficienti i comandi di un comune "linguaggio d'interrogazione".

Come orientarsi nell'acquisto se i record che si intendono creare devono fare riferimento a informazioni diverse da quelle che essi contengono

I DBM più semplici possono gestire un solo file di record alla volta e tale funzionamento è perfettamente adeguato a molte applicazioni. In altre, tuttavia, il DBM deve essere capace di elaborare due o più file contemporaneamente. Un classico esempio è quello di un database per la gestione di magazzino nel quale ogni articolo può avere più fornitori. In tal caso occorrono due file indipendenti: uno per gli "articoli" e l'altro per i "fornitori". È quindi estremamente utile poter eseguire una ricerca su un file e, contemporaneamente, rivolgere domande ad altri file collegati.
Per chiarire il concetto, si immagini di gestire un negozio d'antiquariato e di registrare i "pezzi" su un file MAGAZZINO. Per i fornitori è necessario un secondo file sul quale vengono annotati i mobili che essi ci offrono ma che non abbiamo ancora acquistato. Il campo DESCRIZIONE "MOBILIO, SVEDESE" può risultare vuoto, ma da una ricerca incrociata sul file FORNITORI, è possibile ottenere l'informazione KURT JAKOBSEN ANTIK, GAMLA STAN 56, STOCCOLMA, e i record ad esso abbinati:

```
DESCRIZIONE:    SOFÀ
FABBRICANTE:    ANDERSEN
ANNO:           1824
PREZZO:         4.000.000
```

DESCRIZIONE:	SOFÀ
FABBRICANTE:	GRIMM
ANNO:	1874
PREZZO:	2.900.000

In altre parole, quando un solo file non è sufficiente, è necessario consultare altri file. In teoria è sempre possibile inserire in un particolare file un certo numero di campi di riserva per la registrazione di informazioni supplementari ma, nell'applicazione pratica, sorgono alcuni problemi. Supponiamo che il signor Jakobsen comunichi di aver ricevuto due eleganti sofà del XIX secolo, interessanti per il nostro negozio. È molto più comodo registrare questi mobili nel record JAKOBSEN del file FORNITORI, che non esaminare tutti i campi DESCRIZIONE del file MAGAZZINO e annotarvi i nuovi pezzi.
In conclusione, quando l'archivio da gestire richiede l'impiego di più file contenenti informazioni tra loro collegate, conviene acquistare un DBM multi-file.

I campi dipendenti e quelli calcolati si dimostrano molto utili. Ma in quale misura agevolano le operazioni svolte dal DBM?

Alcuni DBM offrono campi di tipo particolare, detti "dipendenti", che vengono visualizzati (durante l'introduzione dei dati e successivamente) solo se necessario. Ad esempio, un archivio che contiene un campo denominato N. DI FIGLI, non visualizza alcun campo supplementare, se il valore di quel campo è 0. Ma se il valore è 1, mostrerà i campi NOME DEL PRIMO FIGLIO, ETÀ DEL PRIMO FIGLIO e così via. Tuttavia solo in rari casi i campi dipendenti si dimostrano indispensabili. Alcuni DBM permettono di usare i dati registrati in particolari campi come argomenti in operazioni matematiche. Tali sistemi consentono, ad esempio, di moltiplicare il numero degli articoli contenuti in un campo GIACENZE DI MAGAZZINO per la cifra registrata nel campo PREZZO di ciascun record di un file e di sommare tutti i risultati per calcolare il valore totale delle giacenze di magazzino. I campi dipendenti sono quasi sempre abbinati ad un linguaggio di programmazione interno al DBM e si dimostrano estremamente utili.

Quali sono le differenze tra i DBM controllati da menu e quelli controllati da comandi? Come orientarsi nella scelta?

I DBM controllati da menu visualizzano una serie di scelte possibili per ogni fase del lavoro, mentre i DBM pilotati da comandi obbligano l'utente ad imparare l'intero vocabolario dei comandi (che spesso richiedono l'uso combinato del tasto Control e dei tasti letterali). I primi creano meno difficoltà al principiante ma i secondi, una volta imparati i comandi, consentono di lavorare con maggior rapidità, soprattutto se i codici di comando richiamano logicamente le funzioni a cui si riferiscono (ad esempio: S per SEARCH, P per PRINT, ecc.).

Esistono funzioni incorporate nei DBM che verificano e impediscano la registrazione di dati sbagliati?

Un DBM ben progettato deve permettere all'utente di specificare il tipo di dato da inserire in un qualsiasi campo: ciò consente di ridurre gli errori di immissione. Per esempio, il DBM dovrebbe rifiutare automaticamente la data 31/2/85 (perché febbraio non può avere 31 giorni) oppure una registrazione del tipo:

PREZZO: Indicatore di flusso dell'olio

I migliori prodotti software consentono sempre di porre dei limiti al tipo di dati consentiti. D'altra parte, però, è bene evitare una verifica eccessivamente rigida, che rappresenta un ostacolo invece di un ausilio. In ogni caso, prima dell'acquisto, è consigliabile esaminare le funzioni di convalida dei dati offerte dal DBM.

In definitiva, che tipo di DBM conviene acquistare?

Non è possibile rispondere in modo assoluto a questa domanda, perché la scelta di un DBM dipende dall'uso che se ne vuole fare. In linea generale conviene orientarsi verso il software su dischetto, capace di gestire più file e dotato di un proprio linguaggio di programmazione oltre che del normale linguaggio d'interrogazione.



La curvatura del braccio

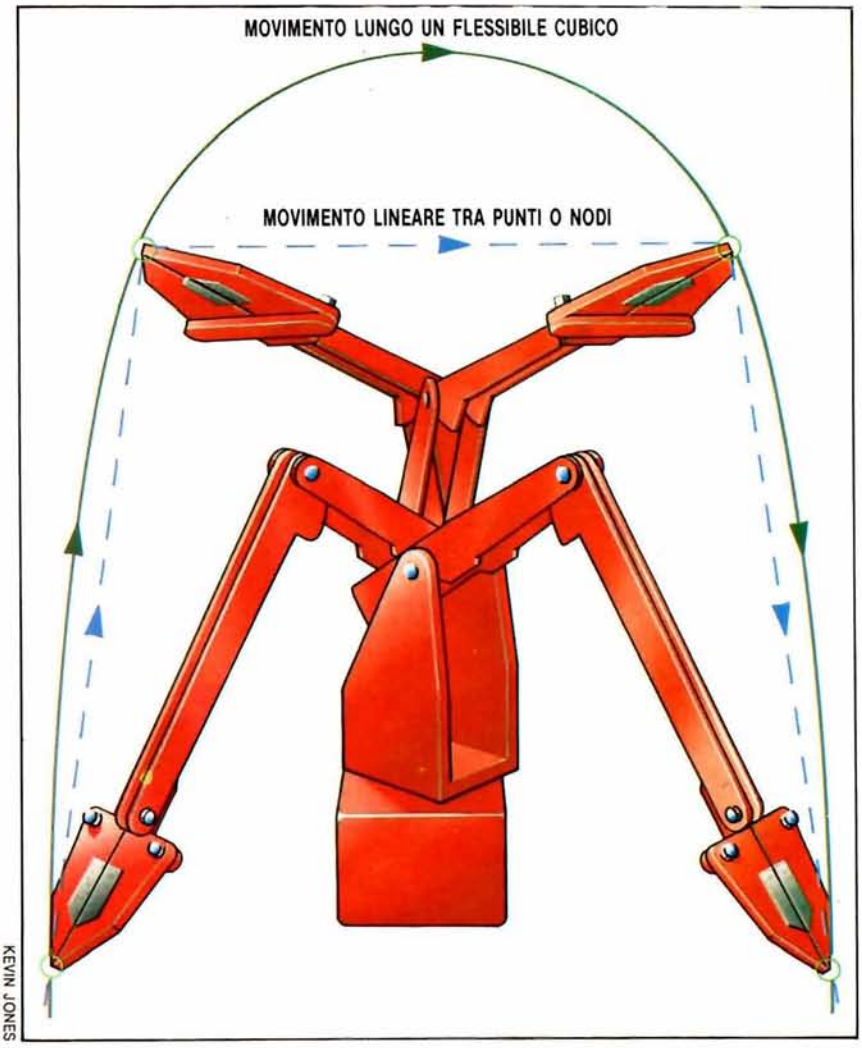
Un ultimo ritocco al software e il braccio è pronto per funzionare

Ottimizzazione del movimento

Si supponga di usare il programma di controllo del robot per salvare quattro posizioni chiave del braccio. Quando la sequenza viene eseguita, il braccio si muove da un punto all'altro in linea retta (linea tratteggiata in figura). È tuttavia possibile calcolare una serie di punti intermedi, appartenenti a una curva (detta "flessibile cubico") che attraversa le quattro posizioni originali. Le coordinate dei punti intermedi possono essere usate per ottimizzare il movimento del braccio

La routine azionaservom, sviluppata per il programma di controllo del braccio robot (vedere alle pagg. 1657 e 1675), riduce i movimenti bruschi eseguiti dal braccio per spostarsi da una posizione alla successiva. Le informazioni per l'angolazione dei motori provengono da una serie di locazioni di memoria che iniziano da ANGOLO.

I motori cambiano angolazione quando viene modificato il contenuto delle locazioni di ANGOLO. Quindi, per armonizzare i movimenti del braccio, i valori non devono essere inseriti direttamente in tali locazioni (con POKE), ma in un gruppo di locazioni equivalenti dette NEWPOS. Quando viene richiamata, la routine azionaservom confronta i valori contenuti nelle locazioni corrispondenti dei due blocchi e incrementa o riduce adeguatamente ANGOLO di un'unità alla volta. La procedura viene ripetuta finché i valori di NEWPOS e ANGOLO non sono identici.



KEVIN JONES

L'introduzione dei valori angolari in NEWPOS, anziché in ANGOLO, permette di compensare efficacemente i cambiamenti di angolazione di maggior ampiezza dei motori e agevola il trasporto di un rocchetto di filo (o di un altro piccolo oggetto) da una posizione data a una scatola. Questa tecnica consente di ottenere spostamenti più lenti del braccio usando i tasti di controllo e di salvare su una matrice le posizioni chiave dell'intero movimento. Tali posizioni possono essere riprodotte passo dopo passo a qualsiasi velocità, introducendo un fattore di ritardo nella routine azionaservom.

Evitare la discontinuità

Il sistema può essere ulteriormente migliorato. In prima approssimazione, tutti i movimenti angolari sono proporzionali ai cambiamenti in NEWPOS e dipendono dal rapporto tra la distanza di ogni raccordo dal perno centrale e il raggio della leva di trasmissione del motore. La routine azionaservom dispone i diversi elementi del braccio nelle nuove posizioni muovendoli scorrevolmente e con velocità angolare (gradi di rotazione per secondo) costante. A questo punto l'intero braccio deve ruotare rapidamente, per assumere la successiva posizione della sequenza. Esiste quindi una discontinuità di movimento.

Questo inconveniente è meno significativo del precedente, ma può essere eliminato salvando una serie di posizioni intermedie tra due posizioni chiave successive.

Occorre perciò realizzare un software più sofisticato che raggiunga un compromesso tra velocità e consumo di memoria. Se la sequenza per spostare il rocchetto di filo prevede solo quattro diverse posizioni chiave (la prima per afferrare il rocchetto, la seconda per sollevarlo, la terza per posizionare il braccio sopra la scatola e la quarta per posare il rocchetto nella scatola), si ottiene un ridotto consumo di memoria ma una notevole discontinuità di movimento che può essere eliminata introducendo 60 posizioni intermedie che rendono il movimento più fluido e omogeneo. Purtroppo questa modifica richiede una maggior quantità di memoria, per non parlare del tempo necessario per digitare tutte le posizioni. L'alternativa implica il calcolo di una serie di valori intermedi che successivamente verranno abbinati alle posizioni registrate manualmente. Vediamo adesso come calcolare questi valori intermedi.

Se si considerano quattro punti, è sempre possibile modellare un flessibile d'acciaio in

modo da collegare tre dei quattro punti, soprattutto se i due punti estremi si trovano in posizione adeguata. L'equivalente matematico di questa operazione è un "flessibile cubico" (così chiamato perché comprende valori elevati alla potenza di 3), che permette l'interpolazione di punti intermedi. Il seguente programma consente di inserire, tra quattro punti dati, una serie di punti intermedi che tracciano una curva cubica. Si ottiene così un ampio movimento curvilineo di gran lunga migliore rispetto a quello che risulta da una semplice interpolazione lineare. Per applicare questa tecnica a un sistema con 4 motori come il nostro braccio robot, occorre interpolare matematicamente un flessibile cubico per ciascun motore.

FLESSIBILI CUBICI

Micro BBC:

```
1000 REM*****************************************
1010 REM ** FLESSIBILI CUBICI PER IL MICRO BBC **
1020 REM*****************************************
1030 :
1040 MODE 0
1050 VDU 23,240,24,126,66,219,219,66,126,24
1060 V=7
1070 DIMX(V+3),Y(V+3),M(V+3),Z(V+3)
1080 X(2)=0 :X(3)=10 :X(4)=40 :X(5)=50
1090 Y(2)=2 :Y(3)=12 :Y(4)=12 :Y(5)=2
1100 REM Prolunga le estremita' in modo lineare
1110 X(1)=X(2)-(X(3)-X(2))
1120 Y(1)=Y(2)-(Y(3)-Y(2))
1130 X(6)=X(5)+(X(5)-X(4))
1140 X(7)=X(6)+(X(6)-X(5))
1150 Y(6)=Y(5)+(Y(5)-Y(4))
1160 Y(7)=Y(6)+(Y(6)-Y(5))
1170 CLG:VDU 5:MOVE 100,700
1180 PRINT "Esempio di routine che usa"
1190 PRINT "un flessibile cubico per"
1200 PRINT "congiungere i punti nodali"
1210 FOR I=2 TO 5
1220 MOVE X(I)*20-4,Y(I)20+15:PRINT CHR$(240);
1230 NEXT
1240 GOSUB 1500: REM SELEZIONA COEFFICIENTI DI AKIMA
1250 FOR X=X(2) TO X(5)
1260 GOSUB 1330: REM FUNZIONE DI AKIMA
1270 IF X=X(2) THEN MOVE X*20,Y*20
1280 IF X<>X(2) THEN DRAW X*20,Y*20
1290 NEXT
1300 END
1310 :
1320 :
1330 REM **** FUNZIONE DI AKIMA ****
1340 REM AKIMA RUCKDESCHEL VOLUME 2 (BYTE/MCGRAW-HILL)
1350 N=1
1360 REM CONTROLLA SE X RIENTRA NEI VALORI IN TABELLA
1370 IF X<X(1) OR X>X(V-2) THEN RETURN
1380 REM CALCOLA L'INTERVALLO RELATIVO ALLA TABELLA
1390 I=0
1400 I=I+1:IF X>=X(I) THEN 1400
1410 I=I-1
1420 REM INIZIA L'INTERPOLAZIONE
1430 B=X(I+1)-X(I)
1440 A=X-X(I)
1450 Y=Y(I)+Z(I)*A+(3*M(I+2)-2*Z(I)-Z(I+1))*A*A/B
1460 Y=Y+(Z(I)+Z(I+1)-2*M(I+2))*A*A*A/(B*B)
1470 RETURN
1480 :
1490 :
1500 REM **** SELEZIONA AKIMA ****
1510 REM CALCOLA I COEFFICIENTI DI AKIMA
1520 FOR I=1 TO V-1
1530 REM SHIFT.I TO I+2
1540 M(I+2)=(Y(I+1)-Y(I))/(X(I+1)-X(I))
1550 NEXT I
1560 M(V+2)=2*M(V+1)-M(V)
1570 M(V+3)=2*M(V+2)-M(V+1)
1580 M(2)=2*M(3)-M(4)
1590 M(1)=2*M(2)-M(3)
1600 FOR I=1 TO V
1610 A=ABS(M(I+3)-M(I+2))
1620 B=ABS(M(I+1)-M(I))
1630 IF A+B<>0 THEN Z(I)=(A*M(I+1)+B*M(I+2))/(A+B)
1640 IF A+B=0 THEN Z(I)=(M(I+2)+M(I+1))/2
1650 NEXT I
1660 RETURN
```

Commodore 64:

```
 10 REM **********************************
 20 REM ** FLESSIBILI CUBICI PER IL C64 **
 30 REM **********************************
 40 :
 50 V=7:FOR I=1 TO 25:DW$=DW$+CHR$(17):NEXT
 60 DIM X(V+3),Y(V+3),M(V+3),Z(V+3)
 70 X(2)=0:X(3)=5:X(4)=20:X(5)=25
 80 Y(2)=2:Y(3)=12:Y(4)=12:Y(5)=2
 90 REM ** PROLUNGA LE ESTREMITA' IN MODO LINEARE **
100 X(1)=X(2)-(X(3)-X(2))
110 Y(1)=Y(2)-(Y(3)-Y(2))
120 X(6)=X(5)+(X(5)-X(4))
130 X(7)=X(6)+(X(6)-X(5))
140 Y(6)=Y(5)+(Y(5)-Y(4))
150 Y(7)=Y(6)+(Y(6)-Y(5))
160 PRINTCHR$(147)
170 PRINT "ESEMPIO DI ROUTINE CHE USA"
180 PRINT "UN FLESSIBILE CUBICO PER"
190 PRINT "CONGIUNGERE I PUNTI NODALI"
200 GOSUB 3000: REM TRACCIA I PUNTI NODALI
230 GOSUB 1000: REM SELEZIONA COEFF. DI AKIMA
235 J=2
240 FOR X=X(2) TO X(5)
250 GOSUB 2000: REM FUNZIONE DI AKIMA
255 IF X=X(J) THEN J=J+1: GOTO 270: REM NESSUNA VISUALIZZAZIONE
260 N=X:M=Y:GOSUB 5000:PRINT CHR$(46)
270 NEXT
280 GET A$:IF A$="" THEN 280
290 END
1000 REM **** SELEZIONA AKIMA ****
1010 REM CALCOLA I COEFFICIENTI DI AKIMA
1020 FOR I=1 TO V-1
1030 M(I+2)=(Y(I+1)-Y(I))/(X(I+1)-X(I))
1040 NEXT I
1050 M(V+2)=2*M(V+1)-M(V)
1060 M(V+3)=2*M(V+2)-M(V+1)
1070 M(2)=2*M(3)-M(4)
1080 M(1)=2*M(2)-M(3)
1090 FOR I=1 TO V
1100 A=ABS(M(I+3)-M(I+2))
1110 B=ABS(M(I+1)-M(I))
1120 IF A+B<>0 THEN Z(I)=(A*M(I+1)+B*M(I+2))/(A+B)
1130 IF A+B=0 THEN Z(I)=(M(I+2)+M(I+1))/2
1140 NEXT I
1150 RETURN
2000 REM **** FUNZIONE DI AKIMA ****
2010 N=1
2020 IF X<X(1) OR X>X(V-2) THEN RETURN: REM CONTROLLA LA GAMMA
2030 I=0
2040 I=I+1:IF X>X(I) THEN 2040
2050 I=I-1
2060 REM INIZIA L'INTERPOLAZIONE
2070 B=X(I+1)-X(I)
2080 A=X-X(I)
2090 Y=Y(I)+Z(I)*A+(3*M(I+2)-Z(I)-Z(I+1))*A*A/B
2100 Y=Y+(Z(I)+Z(I+1)-2*M(I+2))*A*A*A/(B*B)
2110 RETURN
3000 **** VISUALIZZA I PUNTI NODALI ****
3010 FOR I=2 TO 5
3020 N=X(I):M=Y(I): GOSUB 5000: PRINT CHR$(215)
3030 NEXT I
3040 RETURN
5000 REM **** POSIZIONA IN N,M ****
5010 PRINT CHR$(19);
5020 PRINT TAB(N);LEFT$(DW$,25-M);
5030 RETURN
```

Basic & Basic

Spectrum:

Sostituire le linee 1040 e 1050 della versione per il Micro BBC con le seguenti:

1040 CLS

1050 CIRCLE X(I+2)-4, Y(I+2)+15,15

1060 IF X=X(2) THEN PLOT X+8, Y+8

1080 IF X<>X(2) THEN DRAW X+8, Y+8



DATABASE

La sesta tabella della mappa di memoria del Commodore 64. Per concessione della Commodore Business Machines

INDIRIZZO ESADECIMALE	ETICHETTA	LOCAZIONE DECIMALE	DESCRIZIONE
00BF	MYCH	191	**Serial Word Buffer** Buffer per parola seriale
00C0	CAS1	192	**Tape Motor Interlock** Blocco del motore del registratore
00C1-00C2	STAL	193-194	**I/O Start Address** Indirizzo iniziale di ingresso/uscita
00C3-00C4	MEMUSS	195-196	**Tape Load Temps** Locazioni temporanee per il registratore
00C5	LSTX	197	**Current Key Pressed: CHR$(n) 0 = No Key** Tasto attualmente premuto: CHR$(n) oppure 0 = Nessun tasto
00C6	NDX	198	**No. of Chars. in Keyboard Buffer (Queue)** Numero di caratteri nel Buffer della tastiera (coda)
00C7	RVS	199	**Flag: Print Reserve Chars.—1 = Yes, 0 = Not Used** Flag: Visualizz. caratteri inversi: —1 = Si, 0 = Non usato
00C8	INDX	200	**Pointer: End of Logical Line for INPUT** Puntatore: Fine "logica" di una linea in una INPUT
00C9-00CA	LXSP	201-202	**Cursor X-Y Pos. at Start of INPUT** Posizione del cursore all'inizio di una INPUT
00CB	SFDX	203	**Flag: Print Shifted Chars.** Flag: Visualizz. caratteri con shift
00CC	BLNSW	204	**Cursor Blink enable: 0 = Flash Cursor** Abilitazione lampeggio del cursore: 0 = cursore lampeggiante
00CD	BLNCT	205	**Timer: Countdown to Toggle Cursor** Temporizzatore: contatore per scambio di cursore
00CE	GDBLN	206	**Character Under Cursor** Carattere attualmente sotto il cursore
00CF	BLNON	207	**Flag: last Cursor Blink On/Off** Flag: Blink acceso/spento per l'ultimo cursore
00D0	CRSW	208	**Flag: INPUT or GET form Keyboard** Flag: INPUT o GET da tastiera
00D1-00D2	PNT	209-210	**Pointer: Current Screen Line Address** Puntatore: Indirizzo corrente della linea di schermo
00D3	PNTR	211	**Cursor Column on Current Line** Numero della colonna occupata dal cursore nella corrente linea
00D4	QTSW	212	**Flag: Editor in Quote Mode, $00 = NO** Flag: Editor nel modo "virgolette" $00 = NO
00D5	LNMX	213	**Physical Screen Line Lenght** Lunghezza della linea "fisica" sullo schermo

Il Wren Executive

Anatomia di un micro computer a diffusione europea: tecnologie a confronto

Il computer Wren Executive, nonostante la sua "tenera età", ha già una storia piuttosto movimentata.
Costruito dalla Thorn EMI, è stato inizialmente commercializzato dalla Prism e quando, agli inizi del 1985, questa ditta è uscita dal mercato, sembrava che il Wren fosse destinato ad essere un'altra vittima della turbolenta industria dei computer. Invece la distribuzione dell'apparecchio è passata alla Opus Supplies, che ha rilanciato il computer sul mercato.
Il Wren Executive è stato progettato come apparecchio "trasportabile": il contenitore racchiude quindi l'unità centrale, il monitor, le unità a dischetti e la tastiera. Sebbene a prima vista la custodia sembri in metallo verniciato, essa è invece realizzata in un robusto materiale plastico, eccezion fatta per la base della tastiera. La maniglia per il trasporto dell'unità si trova sul lato posteriore dell'apparecchio. Come accade con tanti altri "trasportabili", anche questo è leggermente troppo pesante per essere trasportato a lungo: si tratta, in sostanza, di un apparecchio da tavolo adatto tutt'al più per spostamenti da un ufficio all'altro.
La tastiera ha la solita configurazione QWERTY, con la differenza che sono presenti tasti separati per i simboli * e #. Alla sinistra della tastiera alfabetica ci sono cinque tasti di funzione che, usati assieme ai tasti Shift o Control, forniscono ben 15 diverse combinazioni. La loro funzione dipende dal tipo di applicazione in uso, ma in generale servono per impartire con un tocco solo i comandi del CP/M, quali PIP o RENAME. Dalla parte opposta della tastiera ci sono quattro tasti per il cursore e uno di Home, per far tornare il cursore in alto a sinistra sullo schermo.
Quest'ultimo misura mm. 150 x 105, è monocromatico, ha una risoluzione testuale di 80 x 25 caratteri e non presenta caratteristiche particolari rispetto ad altri monitor del genere. L'unica differenza degna di nota è il colore di primo piano: giallo invece del solito verde. Secondo gli esperti il giallo è un colore che affatica meno gli occhi. Le unità a dischetti da 5 1/4 pollici si trovano alla destra del monitor.
Un inconveniente causato dalla struttura del contenitore è che il monitor, sebbene alloggiato con una certa inclinazione, non si trova sul piano ideale di visualizzazione: può accadere che il bordo della custodia mascheri le prime righe dello schermo.
Sul contenitore non sono stati fissati piedini regolabili (che del resto farebbero inclinare anche la tastiera), quindi l'unica soluzione è adattarsi.
Per quanto concerne le interfacce, il Wren è abbastanza dotato: ci sono varie porte che consentono di collegare il computer ad una vasta gamma di periferiche. Inoltre, diversamente da tanti apparecchi simili, nel Wren è incorporato anche un modem a chiamata automatica, cosicché l'utente può facilmente collegarsi a database pubblici o a reti private senza sobbarcarsi l'acquisto di altre apparecchiature. Viene anche fornito un cavetto di collegamento diretto alle linee telefoniche, ma la presa è di tipo americano, non adatto per l'Italia.



CHRIS STEVENS

UN INSOLITO TRASPORTABILE

Distribuito dalla Opus Supplies, il Wren Executive è un computer maneggevole, che offre un'insolita combinazione: il sistema operativo CP/M e il BASIC versione BBC. Inoltre ha un modem incorporato, due unità a dischetti e un monitor monocromatico

Il software di comunicazione incorporato consente anche la memorizzazione di una certa quantità di numeri telefonici.
Accanto alla porta di comunicazione c'è una coppia di prese per paddle, adatte anche per joystick o mouse. A fianco di queste c'è il connettore D a 25 vie per la porta di comunicazione seriale RS232C.
Normalmente, su altri computer per uso commerciale, questo tipo di porta è usato per il collegamento di modem: sul Wren, essendo questo incorporato, la presa è utile per collegamenti in rete o per l'impiego di stampanti seriali.
A sinistra della porta RS232 c'è un'interfaccia parallela predisposta per il collegamento di un disco fisso tipo Winchester, che può aumentare la capacità della memoria di massa da un minimo di 500 Kbyte fino ad un massimo di 50 Megabyte. Esiste anche un'in-

Abbondanza di interfacce

Una delle caratteristiche più apprezzabili del Wren è la vasta gamma di interfacce offerte di serie: non solo le immancabili (la RS232 seriale e la parallela tipo Centronics), ma anche un connettore per la linea telefonica e una porta parallela per il collegamento di un disco fisso Winchester



Infila la spina e parti!

Il monitor e le unità a dischetti sono protette, durante il trasporto, da una robusta custodia in materiale plastico, nella quale trova alloggiamento anche il cavo di alimentazione. Per lavorare basta rimuovere il coperchio protettivo e collegare il cavo ad una presa di corrente



Parole di perfezione

Poiché il Wren Executive è fabbricato dalla Thorn EMI, è logico che il pacchetto di programmi forniti assieme al computer sia quello commercializzato dalla stessa ditta: il "Perfect Writer", il "Perfect Filer" e il "Perfect Calc". Oltre a questi programmi c'è anche l'"Executive Desktop" (una specie di agenda/calendario, per la gestione di appuntamenti) e il software di comunicazione



Monitor

Sul monitor, monocromatico, le scritte appaiono in giallo anziché in verde, in modo da affaticare meno la vista

Porta RS232C

Questa consente il collegamento diretto con altri computer oppure l'uso di stampanti seriali

Uscita per monitor a colori

Volendo usare un monitor a colori esterno, questa presa fornisce i segnali RGB necessari

Altoparlante

La riproduzione dei suoni generati dal Wren avviene tramite questo altoparlante incorporato

Interfaccia per Winchester

A questa può essere collegata un'unità a disco fisso tipo Winchester, che accresce notevolmente la capacità della memoria di massa

Chip di RAM

Benché il processore Z80 sia ca
di indirizzare al massimo 64 K
mediante un'abile suddivision
banchi il Wren può contare su
400 Kbyte di RAM



terfaccia parallela per stampanti tipo Centronics. Infine, all'estrema destra, c'è una presa per monitor a colori RGB. Tra questa e l'interfaccia parallela è situato un piccolo pulsante per l'avvio "a freddo" del sistema (reset).

Nel Wren viene usato il notissimo processore a 8 bit Z80B. Questo, forse, è l'unico punto debole dell'apparecchio, considerando la netta tendenza della maggioranza dei fabbricanti a produrre computer a 16 bit. Benché l'unità centrale lavori ad una frequenza elevata, la sua velocità non può essere paragonata a quella di un micro a 16 bit, quale l'IBM o il Macintosh.

Avendo scelto come processore lo Z80, è ovvio che il sistema operativo non poteva essere che il famosissimo, ma ormai superato, CP/M. Per facilitarne l'uso, i progettisti lo hanno modificato introducendo un funzionamento tramite menu, che consente di selezionare le varie operazioni semplicemente spostando il cursore e premendo Return. Il sistema invita automaticamente ad inserire il dischetto appropriato e ad avviare la procedura richiesta.

Gli utenti si trovano raramente a dover usare direttamente i comandi del CP/M, poiché la maggior parte delle applicazioni viene avviata tramite i menu del sistema.

Uno dei dischetti forniti assieme al computer contiene oltre al sistema operativo CP/M un word processor (il *Perfect Writer*), un programma per calendario/agenda (l'*Executive Desktop*), un programma per database (il *Perfect Filer*) e, infine, una lavagna elettronica (il *Perfect Calc*). Un secondo dischetto contiene il BASIC e il software di comunicazione.

Il BASIC adoperato sul Wren assomiglia in modo impressionante a quello del Micro BBC e, in effetti, si tratta di una versione riadattata in cui si trovano alcune differenze nelle funzioni di input/output, dovute alla diversità tra il processore 6502 e lo Z80 qui impiegato. Sembra, però, che l'implementazione non sia esente da inconvenienti: nei MODE di visualizzazione 0 e 1, ad esempio, il cursore sparisce inspiegabilmente e il tasto Delete non funziona correttamente (né indietreggiare col cursore e "ripassare" con un nuovo carattere sembra dare risultati soddisfacenti). Tuttavia gli altri modi di visualizzazione paiono funzionare regolarmente.

Il Wren è un computer adatto principalmente ad usi commerciali, ma il suo successo può essere ostacolato dal fatto di essere basato sul processore Z80, ormai sorpassato. Del resto questo apparecchio difficilmente entrerà nel mercato degli home computer, considerata la presenza di apparecchi molto più accattivanti quali l'Amstrad (anch'esso dotato di CP/M) il cui prezzo è sensibilmente inferiore.

Una strada, comunque, rimane aperta ed è quella del settore didattico, poiché il BASIC del BBC è uno tra i più adatti a questo tipo di applicazioni e le caratteristiche hardware del Wren hanno un aspetto più professionale di quelle offerte dagli apparecchi Acorn.

CHRIS STEVENS

WREN EXECUTIVE

DIMENSIONI

mm. 440 x 410 x 250

CPU

Z80B, funzionante a 6 MHz

VIDEO

80 x 25 caratteri in modo testuale. 640 x 256 pixel in modo grafico

INTERFACCE

Porta per monitor a colori, interfaccia Centronics, porta RS232C, presa per collegamento diretto con la linea telefonica, porte per coppia di paddle o joystick, interfaccia per disco fisso tipo "Winchester"

LINGUAGGI DISPONIBILI

BASIC versione BBC

TASTIERA

Tipo QUERTY, con 57 tasti. 5 tasti di funzione e 5 per il controllo del cursore

DOCUMENTAZIONE

Oltre al manuale di sistema, c'è una guida per ogni programma del pacchetto software fornito. Le spiegazioni sono chiare e ben redatte

PRO

Il modem incorporato, la ricchezza di interfacce e il buon numero di programmi forniti assieme all'apparecchio costituiscono un buon punto a favore dell'acquisto

CONTRO

La tecnologia a 8 bit impiegata sul Wren è ormai considerata superata. Inoltre, pare che vi siano alcuni inconvenienti nel funzionamento di qualche programma

Ammutinamento a bordo

Anche questo può accadere nel nostro viaggio verso il Nuovo Mondo

Gli eventi principali e minori possono infuire in senso positivo o negativo sul viaggio. Questi vengono selezionati in modo casuale ed eseguiti ogni settimana, ma non possono verificarsi più di una volta durante la traversata. Esiste tuttavia un evento che, in particolari circostanze, può presentarsi in qualsiasi fase del gioco: se le condizioni di vita a bordo diventano intollerabili, l'equipaggio può ammutinarsi. I fattori che possono provocare un ammutinamento sono 8. Sebbene la nave possieda spazio sufficiente per 16 uomini, qualsiasi numero superiore a 12 crea condizioni di sovraffollamento che rendono turbolento l'equipaggio. Se a bordo non c'è un cuoco (oppure se è morto durante il viaggio), i marinai devono prestare servizio in cucina: ciò riduce la qualità dei cibi e aumenta il nervosismo.
L'avvistamento di un albatro è un segno favorevole (vedere a pag. 1652) e allenta la tensione a bordo. La sua uccisione è invece di cattivo auspicio e rende più probabile un ammutinamento. Il dimezzamento delle razioni settimanali crea ulteriore fermento tra l'equipaggio, perché fa supporre che la cassa di bordo non contenga denaro sufficiente per le paghe. Infine, al momento dell'imbarco, l'equipaggio viene informato che la durata della traversata è di 8 settimane: ogni ritardo crea un malcontento che cresce con l'aumentare delle settimane di viaggio.

Verifica delle condizioni

Il programma usa un "fattore di ammutinamento", MF, per verificare se le condizioni sono tali da scatenare una ribellione. Ogni condizione viene esaminata all'inizio della settimana dalla routine di ammutinamento, chiamata dal ciclo principale del programma. Se il risultato di una verifica è positivo, il fattore di ammutinamento viene incrementato di 1. Quando MF raggiunge 100 si verifica un ammutinamento. La valutazione settimanale del fattore di ammutinamento introduce anche un fattore casuale che può avere un valore massimo di 30.
La linea 879 del programma principale chiama la subroutine alla linea 7200: questa esamina "settimanalmente" il fattore MF, azzerato all'inizio della routine. Se le razioni vengono dimezzate, la variabile H$ assume il

MODULO 10: L'AMMUTINAMENTO

Routine dell'ammutinamento

```
879 GOSUB 7200
```

Da aggiungere al ciclo principale del viaggio

```
7200 REM AMMUTINAMENTO
7210 MF=0
7215 IF H$="S" THEN MF=MF+30
7220 NC=0
7225 FOR T=1 TO 16
7228 IF TS(T,1)=5 AND TS(T,2)<>0 AND TS(T,2)<>-999 THEN
     NC=1: T=16
7230 NEXT
7235 IF NC=0 THEN MF=MF+30
7240 IF A$="S" THEN MF=MF-20
7245 IF B$="S" THEN MF=MF+30
7250 IF CN>12 THEN MF=MF+30
7255 IF WT>MO THEN MF=MF+30
7260 IF WK>8 THEN MF=MF+((WK-8)*10)
7275 MF=MF+INT(RND(1)*30)
7280 IF MF<75 THEN RETURN
7282 PRINT CHR$(147)
7284 IF MF>100 THEN 7300
7285 S$="LE CONDIZIONI DI VITA A*": GOSUB 9100
7286 S$=" BORDO PEGGIORANO *": GOSUB 9100
7287 S$=" E ALCUNI MARINAI*": GOSUB 9100
7288 S$=" TRAMANO UN'AMMUTINAMENTO*": GOSUB 9100
7290 PRINT: GOSUB 9200
7292 S$=K$: GOSUB 9100
7294 GET I$: IF I$="" THEN 7294
7299 RETURN
7300 PRINT CHR$(147)
7305 PRINT: GOSUB 9200
7310 S$="L'EQUIPAGGIO SI E' AMMUTINATO*": GOSUB 9100
7312 S$=" PERCHE':*": GOSUB 9100
7313 X=0
7314 IF H$<>"S" THEN 7320
7315 GOSUB 9200: X=X+1: PRINT X;
7316 S$="LE RAZIONI SONO DIMEZZATE*": GOSUB 9100
7318 S$=" PER PARTE DEL VIAGGIO*": GOSUB 9100
7320 IF NC<>0 THEN 7325
7321 GOSUB 9200: X=X+1: PRINT X;
7322 S$="MANCA IL CUOCO*": GOSUB 9100
7324 S$=" E IL CIBO E' IMMANGIABILE*": GOSUB 9100
7325 IF B$<>"S" THEN 7330
7326 GOSUB 9200: X=X+1: PRINT X;
7327 S$="L'ALBATRO E' STATO UCCISO!*": GOSUB 9100
7330 IF CN<13 THEN 7335
7331 GOSUB 9200: X=X+1: PRINT X;
7332 S$="LA NAVE E' SOVRAFFOLLATA*": GOSUB 9100
7335 IF MO>=WT THEN 7340
7336 GOSUB 9200: X=X+1: PRINT X;
7337 S$="MANCA IL DENARO*": GOSUB 9100
7338 S$=" PER LE PAGHE*": GOSUB 9100
7340 IF WK<=8 THEN 7350
7341 GOSUB 9200: X=X+1: PRINT X;
7342 S$="IL VIAGGIO DURA DA *": GOSUB 9100
7343 S$="PIU' DI 8 SETTIMANE*": GOSUB 9100
7350 PRINT: GOSUB 9200
7360 S$="GLI AMMUTINATI PRENDONO *": GOSUB 9100
7362 GOSUB 9200
7363 S$="LA NAVE E FUGGONO*": GOSUB 9100
7370 GOSUB 9200
7372 S$="ABBANDONANDO IL CAPITANO *": GOSUB 9100
7373 S$="IN MARE APERTO*": GOSUB 9100
7374 S$=" BUON BAGNO!!!!*": GOSUB 9100
7375 PRINT: GOSUB 9200
7380 PRINT"          FINE DEL GIOCO"
7382 END
```

Basic & Basic

Spectrum:

Modifiche per gli altri apparecchi:

```
7282 CLS
7294 LET I$=INKEY$: IF I$="" THEN GO TO 7294
7300 CLS
```

Micro BBC:

```
7282 CLS
7294 I$=GET$
7300 CLS
```

valore S. Poiché l'equipaggio è particolarmente sensibile alle riduzioni di cibo, quando la routine dell'ammutinamento trova S in H$, somma il valore 30 al fattore MF (linea 7215). Quindi la routine verifica se a bordo c'è un cuoco: azzera la variabile NC ed esegue un ciclo (da 1 a 16) che controlla se il primo elemento della matrice tipo/rendimento dell'equipaggio contiene il valore 5 corrispondente al cuoco. Se il rendimento del cuoco è diverso da 0 o −999 (cuoco deceduto), NC viene posta a 1. Se il cuoco non è mai stato imbarcato oppure è morto durante il viaggio, NC rimane a 0 e in tal caso la linea 7235 somma 30 al fattore d'ammutinamento.

Se durante la traversata viene avvistato un albatro, la subroutine dell'albatro assegna S alla variabile A$ (linea 6055) e quella dell'ammutinamento sottrae 20 da MF (linea 7240). Se l'uccello viene ucciso, la linea 6162 assegna S a B$ mentre la linea 7245 incrementa MF di 30.

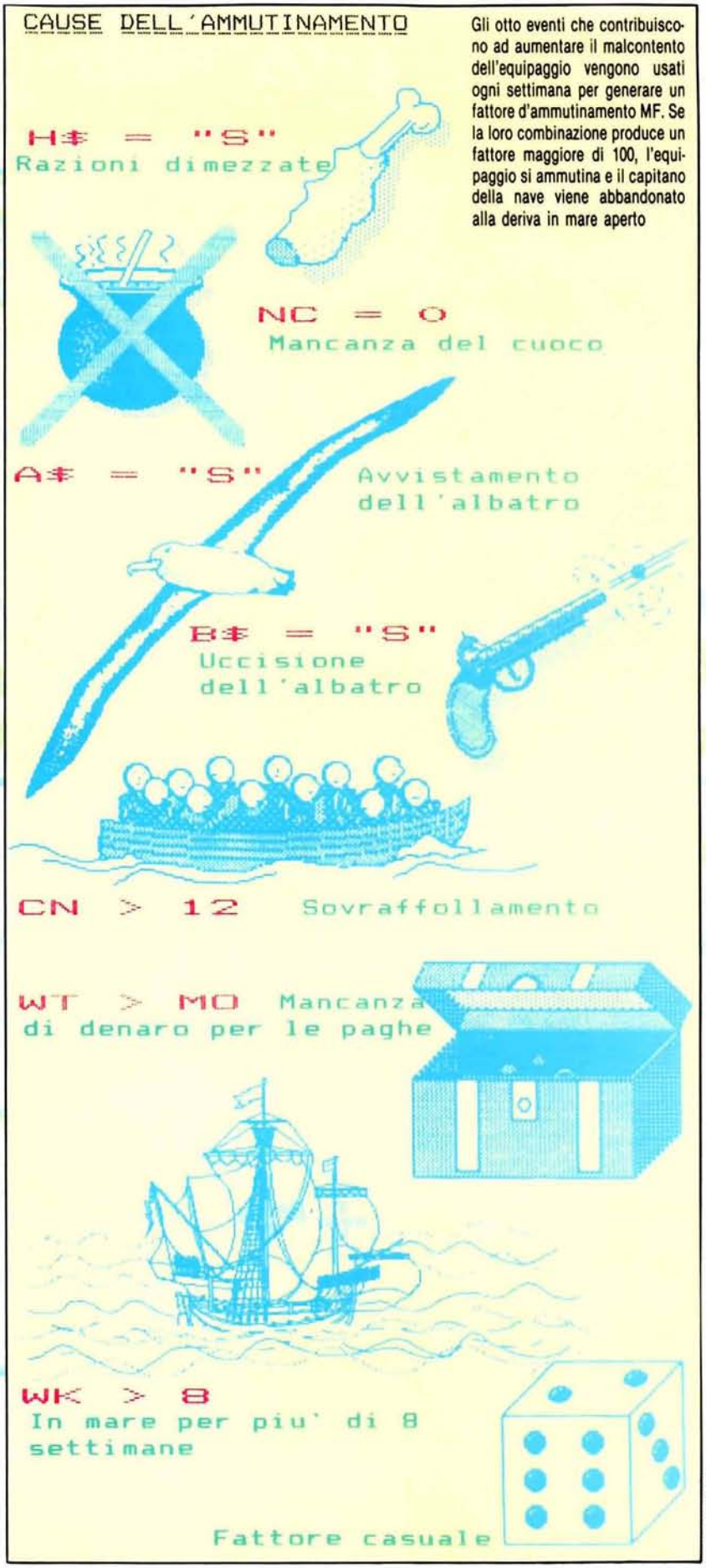
La linea 7250 controlla se i membri dell'equipaggio sono più di 12: in caso affermativo aggiunge 30 al fattore d'ammutinamento per indicare l'eccessivo affollamento della nave.

La linea 7255 verifica se il totale delle paghe (WT) è superiore al denaro rimasto nel forziere del capitano (MO): in tal caso somma 30 a MF. Il fattore d'ammutinamento cresce di 10 per ogni settimana di viaggio supplementare. Se la traversata dura più di 8 settimane, l'equazione alla linea 7269 sottrae 8 dal numero delle settimane (WK), moltiplica il resto per 10 e somma il risultato a MF. La linea 7275 genera un numero casuale compreso tra 0 e 29 e lo aggiunge a MF.

Se, dopo aver verificato tutte le condizioni, il valore del fattore d'ammutinamento è minore di 74, il controllo viene restituito al programma principale (linea 7280). Se invece MF è maggiore di 100, la linea 7284 invia il programma alla linea 7300 e comincia l'ammutinamento. Se MF è compreso tra 75 e 100, le linee 7285-7288, prima di restituire il controllo al programma principale, comunicano al giocatore che la situazione a bordo sta precipitando e che la ciurma trama una rivolta.

Se l'equipaggio si ammutina, il programma ne individua le cause e le visualizza. La linea 7314 verifica se H$ vale S e comunica al giocatore che gli uomini sono stanchi di ricevere razioni dimezzate. La linea 7320 controlla se NC è uguale a 0, ossia se a bordo non c'è un cuoco: in questo caso l'ammutinamento è in parte dovuto alla cattiva qualità dei pasti. La linea 7325 verifica se l'albatro è stato ucciso e informa che tale azione ha esasperato gli animi dei marinai. Le linee 7330 e 7335 controllano rispettivamente se la nave è sovraffollata e se c'è denaro sufficiente per pagare l'equipaggio. Infine la linea 7340 valuta la durata del viaggio.

A questo punto il programma comunica che alcuni marinai sono rimasti uccisi e che gli ammutinati, impadronitisi della nave, fuggono abbandonando il capitano in mare aperto. Poiché l'ammutinamento provoca la fine del gioco, è bene che il giocatore valuti con attenzione le condizioni di vita dell'equipaggio!



Gli otto eventi che contribuiscono ad aumentare il malcontento dell'equipaggio vengono usati ogni settimana per generare un fattore d'ammutinamento MF. Se la loro combinazione produce un fattore maggiore di 100, l'equipaggio si ammutina e il capitano della nave viene abbandonato alla deriva in mare aperto

Un duetto dinamico

L'allocazione dinamica della memoria in Pascal

Simboli usati per i puntatori

Puntatore indefinito (dopo una dichiarazione, prima di un'assegnazione o dopo la chiamata a 'dispose')

Un puntatore 'nullo' (al quale è stato assegnato il valore NIL)



Un 'nodo' di record, in cui un campo contiene un dato e l'altro un puntatore

Il Pascal offre due procedure standard (new e dispose) ed uno speciale tipo di dati, che forniscono una straordinaria potenza di elaborazione nell'allocazione (e de-allocazione) dinamica della memoria. Il tipo "puntatore" è una variabile che, invece di un normale dato (un intero ad esempio), contiene un valore che "punta" ad un dato, sia esso un intero o qualsiasi altro elemento strutturato o no. La notazione è simile a quella impiegata nell'uso dei buffer per l'esame anticipato dei file. Un tipo "puntatore" è definito facendo precedere con una freccia all'insù l'identificatore del dato al quale si vuol puntare:

```
TYPE
    AmpiaMatrice   = ARRAY    1..100 OF real;
    indiretto      = ^AmpiaMatrice;
```

In tutti i compilatori Pascal versione ISO il simbolo @ ha identico significato. La definizione vista riserva una singola locazione di memoria, dove verrà depositato l'indirizzo della matrice, ma soltanto dopo una chiamata alla procedura new. Nel frattempo, il valore del puntatore è indefinito, così come accade normalmente con le variabili. Pertanto:

```
VAR
    indirizzo      :indiretto;
    numero         :integer;
```

riserva spazio in memoria per un indirizzo (16 bit su micro a 8 bit) e per un intero, entrambi dei quali non risultano inizializzati.
Così come accade con le variabili, si può voler inizializzare un puntatore a zero: per far ciò si ricorre alla parola riservata NIL. Azzerare un puntatore ha un significato simbolico, per indicare che il puntatore... non punta proprio a niente! Le due variabili viste si possono quindi azzerare così:

```
indirizzo := NIL
numero := 0;
```

Poiché NIL è un valore costante che appartiene ad un tipo generico, può essere preferibile definirlo come identificatore nel linguaggio e Wirth deve in effetti aver cambiato idea, se nel linguaggio MODULA-2 la parola NIL è definita appunto come un identificatore predefinito e non come parola riservata. Per illustrare l'uso di new e dispose gli esempi adoperano, per semplicità, valori interi.
Quando occorre che un puntatore "punti" a qualche nuovo valore particolare, in PASCAL si adopera la procedura new. Essa non fa che allocare spazio per l'elemento e ne deposita l'indirizzo nel puntatore. Dopodiché, all'elemento si può accedere in via indiretta, tramite il puntatore. Se p è il puntatore, p↑ indica l'elemento puntato. Si noti che la freccia adesso segue l'identificatore (una notazione simile è adoperata nelle operazioni sui buffer dei file). Anche se può sembrare superfluo, occorre ricordare che non ha senso adoperare questa notazione qualora il puntatore non sia inizializzato, oppure l'indirizzo in esso contenuto non si riferisca ad un dato esistente.

De-allocazione dinamica

Una volta terminate le operazioni con i dati indirizzati usando la procedura new, si può invocare la procedura dispose. Questa agisce in modo contrario a new, poiché la memoria riservata con new viene liberata e il puntatore torna a contenere un valore indefinito. Il programma *DuePiuDue* illustra questi eventi:

```
PROGRAM DuePiuDue (output);

TYPE
      puntatore       = ^Integer;

VAR

      p1,
      p2              : puntatore;
      risposta        : integer;

BEGIN
    new (p1);
    p1^ := 2;
    new (p2);
    p2^ := p1
    risposta := p1^ + p2^
    WriteLn (p1^,' + ',p2^,' = ',risposta);
    dispose (p1);
    dispose (p2);
END.
```

Questo modo insolito di sommare due più due evidenzia due punti molto importanti:

- Le variabili dinamiche sono "anonime": nel programma non esiste alcun identificatore di variabile (ad esempio N) che contenga il valore 2. Questo perché ai valori si accede in modo indiretto.
- Dopo la seconda dispose, l'unica porzione di memoria usata è per l'unico valore intero (risposta): nessun dato dinamico esiste più.

Chi ha seguito la serie per il linguaggio Assembly i principi sui quali si basa l'indirizzamento tramite puntatori del PASCAL non dovrebbero risultare difficili poiché le analogie sono molte. Tuttavia esistono alcune differenze nell'indirizzamento indiretto usato da linguaggi quali il PASCAL. In primo luogo non si conosce, né serve conoscere, quali siano gli indirizzi di memoria coinvolti. L'unico "indirizzo assoluto", in PASCAL, è quello di NIL.

Inoltre, si ha la possibilità di usare, liberare e quindi riutilizzare qualsiasi porzione di memoria disponibile, senza doversi preoccupare della sua riorganizzazione (la cosiddetta "raccolta di rifiuti" o "garbage collection"); infatti, queste operazioni sono condotte dallo stesso PASCAL. Al programmatore basta sapere quanta memoria è disponibile in un dato momento: la funzione MemAvail, non-standard, o la Free nella versione ISO-PASCAL della Acorn, informa su quanti byte liberi rimangono.
Un'altra utile estensione è la funzione SizeOf (IdentificatoreDiTipo), che fornisce la lunghezza in byte di qualsiasi particolare tipo.
La RAM utente è suddivisa in due strutture interne: lo "stack" e lo "heap". Lo stack è usato per le chiamate alle funzioni e alle procedure (qui vengono conservati l'indirizzo di chiamata e tutte le variabili locali ed i valori risultanti). Tutti i dati dinamici vengono invece appoggiati sullo heap, che opera in modo simile allo stack, benché non si tratti di una struttura LIFO (Last In First Out, vedere a pag. 1428). In memoria, lo heap cresce nel verso opposto allo stack: l'uno si avvicina all'altro. Un uso indiscriminato di procedure new, abbinato all'impiego di procedure ricursive, potrebbe causare una collisione tra stack e heap. Ciò si evita con:

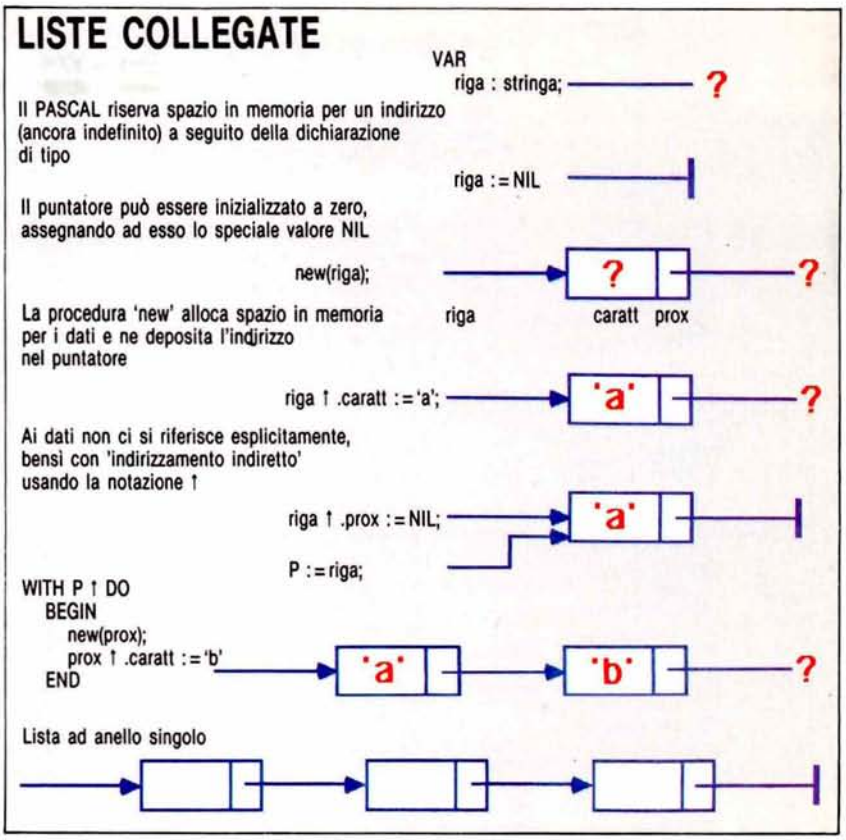
```
IF SizeOf (cosa) > Percentuale * MemAvail THEN...
```

dove cosa è l'identificatore di tipo del dato che si sta per creare e Percentuale è un valore empirico di circa 0.7 (supponendo che il 30% della memoria sia usato per lo stack e il restante 70% per lo heap). In genere, se si adopera MemAvail (o Free) come "segnale di guardia", si usa anche un "contatore di rifiuti" relativo agli elementi non più usati.

Strutture collegate

Dell'effettiva potenza e utilità dei puntatori ci si rende conto nella creazione di strutture collegate quali: alberi, liste concatenate a singolo o doppio elemento, liste circolari, e via dicendo. Si consideri, il problema delle stringhe di caratteri. In genere si adoperano matrici a stringa di dimensione fissa, ad esempio 80 caratteri. Ma usando un tale sistema, si occupano 80 byte anche quando una riga di testo contiene un solo carattere o anche tutti spazi! Al contrario, se occorre una riga più lunga di 80 caratteri, la rigida struttura della matrice non consente di sconfinare.
In PASCAL il problema può essere risolto adoperando un record suddiviso in due campi: uno contiene i dati (una riga di testo nel nostro caso) e viene definito come tipo char, mentre l'altro è costituito da un puntatore al "prossimo" record, posto che ne esista uno.

```
TYPE
     stringa     = ↑carattere;
     carattere   = RECORD
                     caratt : char;
                     prox : stringa
                     END;
```



LIZ DIXON

```
VAR
      riga        : stringa;
BEGIN
   riga := NIL;
        ecc.
```

Una stringa vuota si ottiene semplicemente assegnando il valore NIL alla stringa. Qualsiasi altra sequenza di caratteri implica l'esistenza di un nuovo record per contenere ciascun char e un altro puntatore per il record successivo. Il campo prox, nell'ultimo record, viene inizializzato a NIL, in modo da individuare la fine della stringa. Una procedura per visualizzare la stringa potrebbe essere:

```
WHILE riga <> NIL DO
   BEGIN
      write (riga caratt);
      riga := riga prox
   END
```

La struttura di dati è di tipo ricursivo, poiché è definita in termini di se stessa. Al campo del puntatore non può essere assegnato un tipo completamente definito, così viene creato un riferimento "in avanti".

In successione

Il PASCAL offre la possibilità di implementare con facilità strutture a 'lista collegata', in cui ogni elemento contiene un puntatore all'elemento successivo. Le liste possono essere a collegamento singolo (come quella appena descritta), doppio (ogni elemento contiene due puntatori: uno all'elemento successivo e uno a quello precedente) o, infine, a collegamento circolare (l'ultimo elemento della lista punta al primo)

PROGRAMMA PER LISTA CIRCOLARE

```
PROGRAM   ListaCirc (input, output);

{-------------------------------------------------------
 ---- Scopo: Inserimento di record contenenti dati di tipo
 ---- misto in una lista circolare allocata dinamicamente.
 ---- I dati sono inseriti in ordine alfabetico, secondo
 ---- un   campo  usato  come  chiave   (Nome),   rendendo
 ---- superfluo un algoritmo di riordino.
 -------------------------------------------------------}

CONST
     LunghStringa    = 25;
     spazio          = ' ';

TYPE
     Cardinale       = 0 .. MaxInt;
```

Un girotondo di nomi

Il programma consente di inserire record che contengano dati di tipo misto in una lista circolare allocata dinamicamente. I dati vengono automaticamente inseriti nel corretto ordine alfabetico in base al valore di uno dei campi (quello del Nome), cosicché non occorre applicare un algoritmo di riordino a parte

```
      DimStringa      = 1 .. LunghStringa;
      stringa         = PACKED ARRAY [DimStringa]
                          OF char;

      cosa            = RECORD
                          Nome    : stringa;
                          (... altri campi)
                          somma   : Cardinale
                        END; (cosa)

      puntatore       = ^nodo;

      nodo            = RECORD
                          elem      : cosa;
                          prox      : puntatore
                        END; (nodo)

VAR
      lista           : puntatore;

(11111111111111111111111111111111111111111111111)

PROCEDURE   LeggiCose     ( testa : puntatore);

   VAR
         dato       : cosa;
         OK         : boolean;

   (22222222222222222222222222222222222)

PROCEDURE   SaltaSpazi    (VAR    F : text);
   (ignora gli spazi iniziali)

   VAR
         fatto      : boolean;
   BEGIN
     fatto := EoF ( F );

     WHILE NOT fatto DO
       BEGIN
         fatto := (F^ > spazio) OR EoLn (F);

         IF NOT fatto THEN
           BEGIN
             get (F);
             done := EoF (F)
           END
       END
   END; (SaltaSpazi)

   (22222222222222222222222222222222222)

PROCEDURE  LeggiScheda    (VAR  F : text;
                           VAR  N : Cardinale;
                           VAR OK : boolean);
    (Accetta un numero non-negativo)

   TYPE
       singolo  = 0 .. 9;
   VAR
       cifra    : SET OF char;
       fatto    : boolean;

       (333333333333333333333333333333)

FUNCTION  Valore (cifra   : char)     : singolo;
    (converte carattere in valore numerico)

    BEGIN
       Valore := ord (cifra) - ord ('0')

    END;   (Valore)

       (333333333333333333333333333333)

FUNCTION   Valido (N      : Cardinale;
                   cifra : char)  : boolean;
     (il numero supera MaxInt?)
   BEGIN
       IF N = MaxInt DIV 10
          THEN
             Valido := Valore (cifra) <= MaxInt MOD 10
          ELSE
             Valido := N < MaxInt DIV 10

   END;    (Valido)

       (333333333333333333333333333333)

BEGIN (LeggiScheda)

    cifre := [ '0' .. '9' ];
    OK := NOT EoF (F);

    IF OK THEN (trovata una cifra?)
       OK := F^ IN cifre;

    N := 0;
    fatto := NOT OK;

    WHILE OK AND NOT fatto DO
       BEGIN (costruisce il numero N in base 10)
         N := 10 * N + Valore (F^);
         get (F);
         fatto := Eof (F);

         IF NOT fatto THEN
           fatto := NOT (F^ IN cifre);

         IF NOT fatto THEN
           OK := Valido (N, F^)
       END;
    END;   (LeggiScheda)

   (22222222222222222222222222222222222)

PROCEDURE LeggiRiga  (VAR S  : string);
     (legge una serie di caratteri da tastiera)
   VAR
         indice     : 0 .. LunghStringa;
         simbolo    : char;

   BEGIN
      SaltaSpazi (input);
      indice := 0;

      WHILE NOT EoLn AND (indice < LunghStringa) DO
         BEGIN (mette un altro carattere nella stringa)
           indice := succ (indice);
           read (simbolo);
           S [ indice ] := simbolo
         END;
      IF indice < LunghStringa
         THEN (riempie il resto con NUL)
           FOR indice := indice + 1 TO LunghStringa DO
               S [indice] := chr (0)
         ELSE
           IF NOT EoLn THEN (troppi caratteri)
             WriteLn ( 'ATTENZIONE: immissione troncata');

      ReadLn

   END;  (LeggiRiga)

   (22222222222222222222222222222222222)

PROCEDURE   Inserimento  (lista         :puntatore;
                          dato          :cosa);

   VAR
       appogg,
       anello       : puntatore;
       alfa         : string;
   BEGIN (mette dati nella testa del nodo)

     lista ^ elem := dato;
     appogg := lista;
     alfa   := appogg ^ . prox . elem . Nome;

     WHILE dato . Nome < alfa DO
        BEGIN (si muove nella lista)
          appogg := appogg ^ . prox;
          alfa   := appogg ^ . prox ^ . elem . Nome
        END;
         (nessuna verifica per duplicati)
     new (anello);                (crea un altro nodo)
     anello ^ . elem := dato;     (inserisce il dato)
     anello ^ . prox := appogg ^ . prox; (e l'anello)
     appogg ^ . prox := anello           (nella lista)

   END;  (Inserimento)

   (22222222222222222222211111111111111111111)

BEGIN  (LeggiCose)

     write ('Nome ? ');
     ReadLn (dato . Nome);
        (fine con immissione nulla)
     WHILE dato . Nome [ 1 ] <> chr (0) DO
       BEGIN
         REPEAT
           REPEAT
             write ('Ammontare ? ' : 20);

             IF EoLn (input) THEN
                ReadLn (input);

             SaltaSpazi (input)

           UNTIL NOT EoLn (input);

           LeggiScheda (input, dato . somma . OK);

           IF NOT OK THEN (nessuna cifra)
             WriteLn ('*** ERRORE -' : 20,
                      'reimmettere ***')

         UNTIL OK;

         Inserimento (testa, dato);
         WriteLn ( '(terminare con RETURN)' : 40);
         write ( 'Nome ?');
         LeggiRiga (dato . Nome)

       END; (LeggiCose)

  (11111111111111111111111111111111111111111111111)

PROCEDURE Visualizza (lista  :puntatore);

   VAR
         visual   : puntatore;
   BEGIN (prima salta la testa del nodo fasulla)
       visual  := lista ^ . prox;
       page (output);
       WriteLn ('Lista Ordinata:' );
       WriteLn;
         (altri dati da visualizzare?)
       WHILE NOT (visual = lista ) DO
          BEGIN
            WITH visual ^ . elem DO
              WriteLn (somma : 8, Nome : 300;
            WriteLn;
            visual := visual ^ . prox
          END
   END;     (Visualizza)

  (11111111111111111111111111111111111111111111111)

BEGIN    (ListaCirc - Programma Principale)

     new (lista);   (crea una testa di nodo fasulla)
     lista ^ . prox := lista;   (punta su se stesso)
     page (output);
     WriteLn ('Immettere i dati: Prima il cognome, poi' );
     WriteLn ('la somma in Lire :');
     WriteLn;
     LeggiCose (lista);  (immette i dati nella lista)
     Visualizza (lista); (adesso visualizza in ordine)
     WriteLn ('--- fatto ---' : 40)

END.
```

Il "beat" dei byte

Come programmare il Sound Interface Chip (SID) del Commodore 64

Di solito i suoni prodotti dal Commodore 64 vengono inviati direttamente al televisore attraverso l'interfaccia ad alta frequenza, ma si può anche collegare l'uscita audio a un sistema hi-fi per ottenere riproduzioni di alta qualità. Esaminiamo quindi i principi coinvolti nella progettazione di un software che permetta di trasformare il Commodore 64 in una "batteria elettronica". Ovviamente la qualità dei risultati è migliore se viene impiegato un sistema di riproduzione ad alta fedeltà.

Oltre a produrre sofisticati suoni sotto il controllo del software, il chip SID può ricevere segnali audio generati da una periferica elettronica (eventualmente dotata di altri chip SID) o da strumenti musicali come chitarre elettriche. Il segnale ricevuto può essere "mixato" con l'uscita audio del chip SID e elaborato dai suoi filtri. Le interfacce di I/O, però, vanno adoperate con estrema attenzione poiché un errato collegamento delle linee esterne può danneggiare gravemente il computer. Prima di realizzare qualsiasi collegamento esterno, è consigliabile consultare sul manuale tecnico i dati relativi ai piedini e ai livelli di corrente.

Per introdurre l'argomento, vediamo come sia possibile generare un suono musicale periodico, impiegando una serie di note pure; in seguito viene esaminato un metodo alternativo per ottenere lo stesso risultato. Controllando l'inviluppo di una particolare nota periodica, è possibile, in diverse proporzioni, introdurre delle "armoniche" nel suono. Ciò significa che si può creare qualsiasi suono periodico intervenendo su un ristretto numero di controlli relativi all'inviluppo di un oscillatore. Questi sono noti anche sotto la sigla "ADSR" (vedere alle pagg. 276 e 473).

Sfortunatamente il BASIC del Commodore 64 non consente un sofisticato controllo dei suoni che vanno quindi programmati con un massiccio impiego di istruzioni PEEK e POKE. Se si assegna alla variabile SID il valore 54272 (l'indirizzo di base del chip SID), allora gli indirizzi compresi tra SID e SID+28 controlleranno il chip di suono e, di conseguenza, tutti i suoni generati sul Commodore 64.

Infine, il programma in codice macchina a pagina 1720 trasforma il computer in una batteria elettronica a tre voci controllata da interrupt. Questa routine di "cuneo" in codice macchina può essere controllata dal BASIC, ma indipendentemente da questo, continua a suonare il ritmo. Quindi, con opportune modifiche, è possibile abbinarlo a qualsiasi programma BASIC per ottenere effetti sonori di sottofondo.

Suoni musicali

I suoni raggiungono l'orecchio umano come vibrazioni periodiche della pressione atmosferica. Il numero di vibrazioni per secondo è detto "altezza" o "frequenza" del suono. Il limite inferiore dell'udito umano è di circa 15 cicli al secondo (ossia 15 Hertz): una nota pura con frequenza di 100 Hz, quindi, è appena percettibile. Il LA superiore al DO centrale ha, per convenzione, una frequenza di 440 Hz. Se si raddoppia la frequenza di una nota, la sua altezza s'innalza di un'ottava. Mediamente l'orecchio umano percepisce una gamma di suoni dell'estensione di circa 10 ottave. Gli oscillatori a tre note del SID, invece, possiedono una gamma di circa 8 ottave, approssimativamente da 0 a 4000 Hz.

Il fisico francese Jean Fourier (1768-1830) per primo osservò che qualsiasi forma d'onda periodica può essere considerata come composta da una nota pura fondamentale, amalgamata con note le cui frequenze sono multipli della nota fondamentale: tali note prendono il nome di "armoniche". Il timbro inconfondibile di un suono o di una nota dipende dunque dalle proporzioni tra le diverse frequenze armoniche che la compongono.

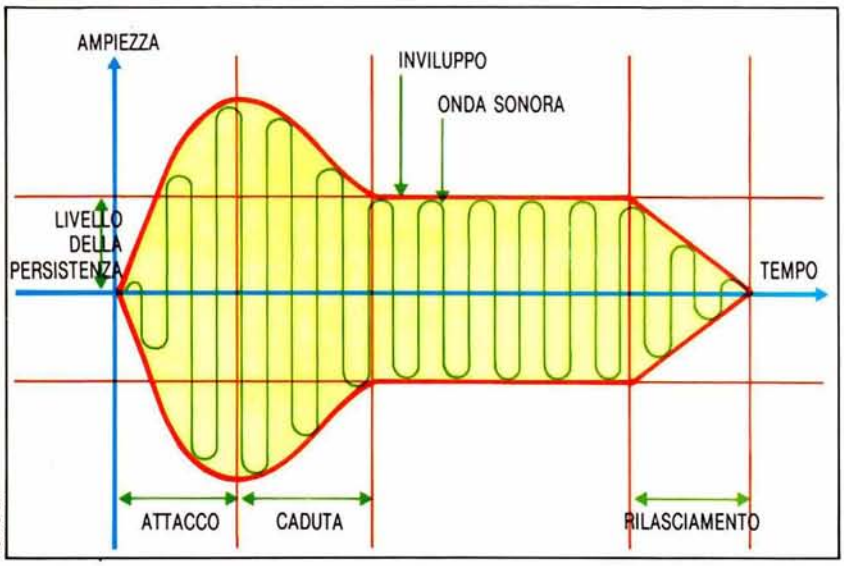
Un'onda sinusoidale pura, che corrisponde a una nota pura, è essenzialmente un segnale analogico difficilmente riproducibile con uno strumento digitale in cui sono presenti livelli di tensione di 0 o 5 volt. Quindi, la tecnica adottata sui micro per ottenere un particolare suono periodico non prevede la generazione di frequenze pure e la loro successiva fusione. Ciascuna delle tre voci generabili dal chip SID può produrre le seguenti 4 forme d'onda periodiche:

1) *Dente di sega:* in essa sono presenti tutte le armoniche. L'armonica "ennesima" possiede un'intensità proporzionale a 1/N.
2) *Triangolare:* contiene solo le armoniche dispari. Per N dispari, l'ennesima armonica possiede un'intensità proporzionale a $1/N^2$.
3) *Quadra:* quest'onda presenta armoniche dispari proporzionali a 1/N. Variando l'ampiezza dell'impulso si possono generare svariate onde rettangolari, ciascuna composta da un proprio amalgama di armoniche.
4) *Rumore bianco:* mistura casuale di frequenze, usata specie per effetti particolari.

Il contenuto armonico di un suono può essere modificato mediante l'impiego dei filtri. Il chip SID offre tre tipi di filtro: passa-basso, passa-banda e passa-alto. Il primo, ad esempio, lascia passare tutte le frequenze minori di un valore prestabilito e attenua tutte le

frequenze superiori ad esso. Grazie a queste funzioni e ai controlli d'inviluppo ADSR è possibile ottenere qualsiasi suono, anche una simulazione del parlato.
I programmi per la generazione del parlato si possono ottenere in più modi e qui ne viene descritto uno che abbina una discreta qualità a un vocabolario illimitato. I suoi elementi di base o "fonemi" vengono codificati in una matrice di valori ADSR. Un programma in codice macchina permette di tradurre i testi ASC (il codice alternativo all'ASCII del Commodore) in una sequenza di codici di fonema, successivamente inviati al chip SID per mezzo della matrice ADSR. Tale operazione è meno semplice di quanto possa sembrare, poiché le norme che regolano la trasposizione di un testo in fonemi sono piuttosto complesse. La qualità del parlato finale dipende principalmente dalla qualità di questa sezione del programma. Comunque, questa applicazione è realizzabile sul Commodore 64 e il mercato offre già numerosi pacchetti commerciali basati su di essa.

Controllo dell'inviluppo



Inviluppi indirizzati

La qualità di una nota, ossia l'insieme delle caratteristiche che permettono di distinguere, ad esempio, il suono di un pianoforte da quello di un violino, dipende dalla forma o "profilo" dell'inviluppo. Nella sintetizzazione elettronica dei suoni, l'inviluppo si compone di quattro diverse fasi dette: Attacco, Caduta, Persistenza e Rilasciamento (ADSR, da Attack, Decay, Sustain e Release). La durata di ciascuna fase viene programmata inserendo (con POKE) particolari valori nei registri del chip SID. Questa tecnica consente di sintetizzare sul Commodore 64 i suoni di diversi strumenti

Il precedente diagramma ADSR rappresenta la forma d'onda generica di una nota musicale e ne evidenzia le caratteristiche controllabili con il chip SID. I fattori ADSR sono:

1) *Attacco:* il tempo di salita di una nota.
2) *Caduta:* il tempo che una nota impiega a ricadere a un livello stabile.
3) *Persistenza:* il volume durante un livello stabile.
4) *Rilasciamento:* il tempo necessario affinché il volume di una nota passi dal livello di persistenza a 0.

Le sezioni di Attacco, Caduta e Rilasciamento di una nota vengono controllate da "nybble" (4 bit) interni ai registri del chip SID. Quindi ciascuno di questi parametri può assumere solo valori compresi tra 0 e 15 e il rapporto tra i loro valori deve essere inserito (con POKE) nei registri del SID. La seguente tabella contiene gli effettivi valori di sincronizzazione:

Val.	Cadenza di Attacco	Cadenza di Caduta Persistenza
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 sec
11	800 ms	2.4 sec
12	1 sec	3 sec
13	3 sec	9 sec
14	5 sec	15 sec
15	8 sec	24 sec

La durata della fase di Persistenza dipende da un ciclo di ritardo. In base alla precedente tabella, i valori ADSR per una nota di violino dovrebbero essere:

ADSR	Tempo	Valori POKE
A	500 ms.	10
D	300 ms.	8
S	—	—
R	750 ms.	9

Per produrre un suono sul Commodore 64, occorre programmare 6 operazioni diverse:

1) Accendere il volume con l'istruzione:
   POKE SID+24,15
2) Selezionare l'ADSR. Ad esempio:
   POKE SID+5,9: REM ATTACCO/CADUTA VOCE #1
   POKE SID+6,0: REM PERSISTENZA/RILASCIAMENTO VOCE #1
3) Selezionare la frequenza di ogni oscillatore. Ad esempio:
   POKE SID+1,25: REM BYTE-ALTO FREQUENZA VOCE #1
   POKE SID,0 : REM BYTE-BASSO FREQUENZA VOCE #1
4) Selezionare la forma d'onda prescelta. Ad esempio:
   POKE SID+4,33: REM SELEZIONA L'ONDA A DENTE DI SEGA PER LA VOCE #1

A questo punto inizia l'emissione del suono, detta anche "apertura della porta".

5) Esecuzione di un ciclo di ritardo mentre la nota si trova a livello di Persistenza.
6) Rilasciamento della forma d'onda. Ad esempio:
   POKE SID+4,32: REM RILASCIAMENTO DELL'ONDA A DENTE DI SEGA

Il modo più semplice per programmare l'emissione di una nota sul Commodore 64 è selezionare i valori iniziali di ADSR e costruire un ciclo FOR che legga (READ) da frasi DATA i byte alto/basso della frequenza. La presenza di valori 0 in queste istruzioni permette di modificare il ritmo delle diverse voci, mantenendo costante la durata del ritardo.

BATTERIA ELETTRONICA

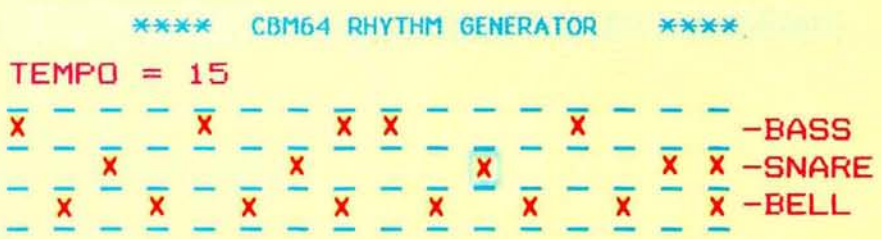
Il seguente programma usa un "cuneo" IRQ in codice macchina, che continua a generare il suono di una batteria indipendentemente dal lavoro svolto dal programma BASIC. Inoltre la sezione in codice macchina esamina, per individuare quali tasti sono stati battuti, il contenuto della locazione 197 ($00C5). Questa è una locazione di pagina 0 che contiene il valore relativo all'ultimo tasto premuto. Al posto di una chiamata al kernel può essere adoperato un programma in codice macchina per la scansione della tastiera, a patto che $00C5 venga esaminato con sufficiente frequenza.

Poiché il programma pilota in BASIC non usa le istruzioni GET o INPUT per svuotare il buffer della tastiera, la routine in codice macchina deve, come ultima operazione, azzerare il puntatore nella locazione 198 ($00C6). Questa è la locazione di pagina 0 che abitualmente contiene il numero dei tasti battuti e accumulati nel buffer della tastiera.

Durante l'esecuzione del programma, ad ogni tamburo vengono assegnati 16 "intervalli" di tempo, visualizzati sullo schermo, durante i quali esso può suonare o restare muto. La scelta del valore 16 facilita la composizione di ritmi rock in quattro quarti. Lo sprite del cursore viene quindi spostato sulla griglia degli intervalli e posizionato sulla cella desiderata; a questo punto basta premere il tasto Return per attivare o disattivare un intervallo di tempo. Per avviare il ritmo selezionato si preme il tasto f1. Il menu sullo schermo visualizza le scelte possibili. Poiché l'emissione dei suoni è controllata da interrupt, la combinazione ritmica può essere modificata sul video mentre il ritmo viene suonato.

Il programma BASIC contiene tutto il codice macchina necessario (sotto forma di frasi DATA) per far funzionare il programma e può essere trascritto ed eseguito senza alcuna modifica. Se si desidera utilizzare il listato in Assembly della parte del programma in codice macchina (vedere alla pagina seguente), occorre assemblarlo e poi eliminare dal programma BASIC le linee da 1520 a 1540 e le istruzioni DATA tra le linee 1620 e 1940.

Dopo aver lanciato il programma, si provi a selezionare la seguente combinazione, che genera un ritmo facilmente riconoscibile:



Programma BASIC generatore di ritmo

```
1000 REM **GENERATORE DI RITMO **
1010 PRINT CHR$(147): REM RIPULISCE LO SCHERMO
1020 GOSUB 1510:REM PREPARA COD. MACCHINA & SPRITE
1030 GOSUB 1270:REM PREPARA LO SCHERMO
1040 SYS(49152):REM *INSERISCE IL CUNEO*
1050 REM SYS(49175)PER RIMUOVERE IL CUNEO
1060 REM *CICLO PRINCIPALE*
1070 P=PEEK(197):REM ULTIMO TASTO BATTUTO
1080 IF P=37 THEN Y=Y-16: SY=SY-1: IF Y<95 THEN Y=95:
     SY=0
1090 IF P=36 THEN Y=Y+16: SY=SY+1: IF Y>127 THEN Y=127:
     SY=2
1100 IF P=47 THEN X=X-16: SX=SX-1: IF X<28 THEN X=28:
     SX=0
1110 IF P=44 THEN X=X+16: SX=SX+1: IF X>268 THEN X=268:
     SX=15
1120 Z=0: IF P=1 THEN Z=1
1130 PRINT CHR$(19); S$;"TEMPO="; PEEK(679); CHR$(157);
     "  "
1140 POKE VIC+16,X/256: POKE VIC,X AND 255
1150 POKE VIC+1,Y
1160 REM *CALCOLA LA POSIZIONE DI SCHERMO IN BASE ALLE
     COORDINATE X,Y DELLO SPRITE
1170 SC=B+INT((Y-50)/8+1)*40+INT((X-18)/8)
1180 IF SY=0 THEN ROW=50000:REM TAMBURO BASSO
1190 IF SY=1 THEN ROW=50016:REM TAMBURO RULLANTE
1200 IF SY=2 THEN ROW=50032:REM CAMPANA
1210 IF Z=1 AND PEEK(SC)=32 THEN POKE SC,24: POKEROW+SX,1:
     GOTO 1230
1220 IF Z=1 AND PEEK(SC)=24 THEN POKE SC,32: POKEROW+SX,0
1230 IF P=51 THEN GOSUB 1270: FOR I=0 TO 50: POKE 50000+I,0
     : NEXT
1240 IF P=57 THEN POKE 198,0: POKE VIC+21,0: PRINT CHR$
     (147): END
1250 GOTO 1070
1270 PRINT CHR$(19)
1260 REM *PREPARA LO SCHERMO*
1280 S$=CHR$(17)+CHR$(17)+CHR$(17)
1290 PRINT TAB(3)"                                        "
1300 PRINT CHR$(18)TAB(3)" **** GENERATORE DI RITMO CBM 64 **** "
1310 PRINT: PRINT: PRINT
1320 PRINT" - - - - - - - - - - - - - - - - - "
1330 PRINT"                                    -BASSO"
1340 PRINT" - - - - - - - - - - - - - - - - - "
1350 PRINT"                                   -RULLANTE"
1360 PRINT" - - - - - - - - - - - - - - - - - "
1370 PRINT"                                   -CAMPANA"
1380 PRINT" - - - - - - - - - - - - - - - - - "
1390 PRINT: PRINT TAB(8)"OPZIONI....."
1400 PRINT TAB(8)"[F1]     SUONA UN RITMO"
1410 PRINT TAB(8)"[F3]     RIDUCE IL TEMPO"
1420 PRINT TAB(8)"[F5]     AUMENTA IL TEMPO"

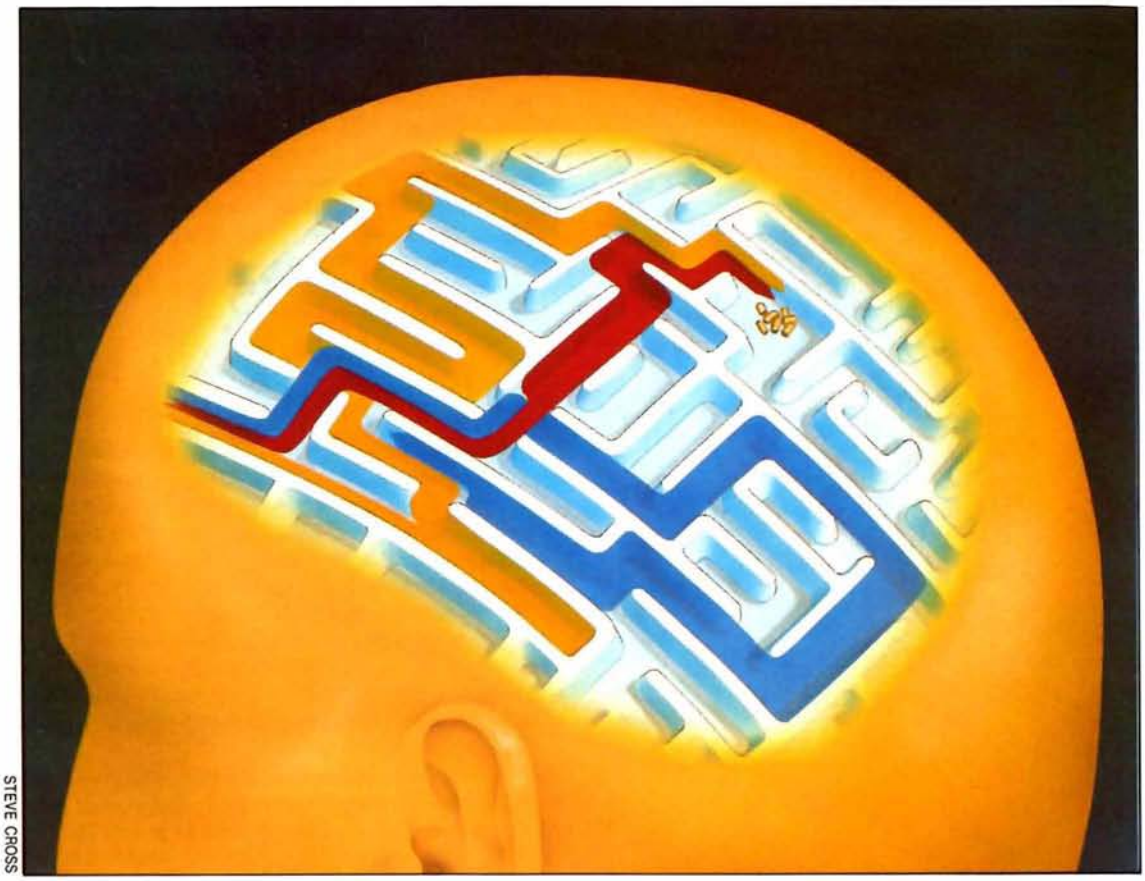
1430 PRINT TAB(8)"[F7]     FERMA IL RITMO"
1440 PRINT TAB(8)"[CLR]    CANCELLA LA GRIGLIA"
1450 PRINT TAB(8)"[8T]     FINE DEL PROGRAMMA"
1460 PRINT TAB(8)"[<]      CURSORE A SINISTRA"
1470 PRINT TAB(8)"[>]      CURSORE A DESTRA"
1480 PRINT TAB(8)"[K]      CURSORE IN ALTO"
1490 PRINT TAB(8)"[M]      CURSORE IN BASSO"
1500 RETURN
1510 REM *PREPARA COD. MACCHINA & SPRITE*
1520 FOR I=49152 TO 49413
1530 READ J: C=C+J: POKE I,J: NEXT I
1540 READ J: IF C<>J THEN PRINT "DATA ERROR": END
1550 FOR I=0 TO 62: READ J: POKE 832+I,J: NEXT I
1560 VIC=53248: X=28: Y=95: B=1024: SID=54272
1570 FOR I=0 TO 24: POKE SID+I,0: NEXT I
1580 POKE SID+24,15: POKE VIC+21,1: POKE 2040,13
1590 POKE 254,15: POKE 679,15: POKE VIC+39,1
1600 FOR I=0 TO 50: POKE 50000+I,0: NEXT
1610 RETURN
1620 REM ***ISTRUZIONI DATA DEL COD MACCHINA***
1630 DATA120,173,20,3,133,251,173,21,3
1640 DATA133,252,169,36,141,20,3,169
1650 DATA192,141,21,3,88,96,120,165,251
1660 DATA141,20,3,165,252,141,21,3,88
1670 DATA96,32,177,192,165,197,201,3
1680 DATA208,3,32,135,192,165,197,201,4
1690 DATA208,3,32,140,192,165,197,201,5
1700 DATA208,3,32,155,192,165,197,201,6
1710 DATA208,3,32,166,192,32,87,192,142
1720 DATA168,2,140,169,2,76,49,234,165
1730 DATA253,208,1,96,198,254,240,1,96
1740 DATA32,149,192,224,16,208,3,32,144
1750 DATA192,185,80,195,240,3,32,184
1760 DATA192,185,96,195,240,3,32,210
1770 DATA192,185,112,195,240,3,32,236
1780 DATA192,200,234,232,96,169,0,133
1790 DATA253,96,169,1,133,253,162,0,160
1800 DATA0,96,173,167,2,133,254,96,173
1810 DATA167,2,201,255,240,3,238,167,2
1820 DATA96,173,167,2,201,1,240,3,206
1830 DATA167,2,96,174,168,2,172,169,2
1840 DATA96,169,14,141,6,212,169,32,141
1850 DATA2,212,169,66,141,4,212,169,3
1860 DATA141,1,212,169,65,141,4,212,96
1870 DATA169,7,141,12,212,169,12,141,13
1880 DATA212,169,128,141,11,212,169,65
1890 DATA141,8,212,169,129,141,11,212
1900 DATA96,169,2,141,19,212,169,13,141
1910 DATA20,212,169,18,141,18,212,169
1920 DATA100,141,15,212,169,17,141,18
1930 DATA212,96
1940 DATA32038:REM*CHECKSUM*
1950 REM ***ISTRUZIONI DATA DELLO SPRITE (CURSORE)***
1960 DATA127,254,0,127,254,0,127,254,0
1970 DATA112,14,0,112,14,0,112,14,0,112
1980 DATA14,0,112,14,0,112,14,0,112,14
1990 DATA0,112,14,0,127,254,0,127,254,0
2000 DATA127,254,0,0,0,0,0,0,0,0,0,0,0
2010 DATA0,0,0,0,0,0,0,0,0,0,0
```

LISTATO ASSEMBLY

```
;++++++++++++++++++++++++++++++++
;++++++++++++++++++++++++++++++++
;++                            ++
;++      CODICE SORGENTE       ++
;++ PER BATTERIA  ELETTRONICA ++
;++                            ++
;++++++++++++++++++++++++++++++++
;++++++++++++++++++++++++++++++++
;
VOL       =  $D418    ;SID VOLUME
ATT1      =  $D405    ;SID ATTACCO VOCE 1
PERS1     =  $D406    ;SID PERSISTENZA VOCE 1
IMPULSO   =  $D402    ;SID FREQ. D'IMPULSO VOCE 1
ONDA1     =  $D404    ;SID FORMA D'ONDA VOCE 1
BASSO     =  $D401    ;SID BYTE-ALTO DELLA FREQ. VOCE 1
ATT2      =  $D40C    ;SID ATTACCO VOCE 2
PERS2     =  $D40D    ;SID PERSISTENZA VOCE 2
ONDA2     =  $D40B    ;SID FORMA D'ONDA VOCE 2
RULLANTE  =  $D408    ;SID BYTE-ALTO DELLA FREQ. VOCE 2
ATT3      =  $D413    ;SID ATTACCO VOCE 3
PERS3     =  $D414    ;SID PERSISTENZA VOCE 3
ONDA3     =  $D412    ;SID FORMA D'ONDA VOCE 3
CAMPANA   =  $D40F    ;SID BYTE-ALTO DELLA FREQ. VOCE 3
RIGA1     =  $C350    ;DEPOSITO PER VOCE 1
RIGA2     =  $C360    ;DEPOSITO PER VOCE 2
RIGA3     =  $C370    ;DEPOSITO PER VOCE 3
TEMPO     =  $02A7    ;DEPOSITO TEMPORANEO PER IL RITARDO
XCONT     =  $02A8    ;DEPOSITO TEMPORANEO PER REG. X
YCONT     =  $02A9    ;DEPOSITO TEMPORANEO PER REG. Y
BBVETT    =  $FB      ;DEPOSITO PER BYTE-BASSO VETTORE
BAVETT    =  $FC      ;DEPOSITO PER BYTE-ALTO VETTORE
SUONA     =  $FD      ;INNESCO PER EMISSIONE DEL SUONO (1=SI)
RITARDO   =  $FE      ;DEPOSITO PER RITARDO CORRENTE
TASTO     =  $C5      ;ULTIMO TASTO PREMUTO
;-------------------------------------------------------------
*         =  $C000    ;ASSEMBLA DA 49152 (DECIMALE)
;-------------------------------------------------------------
;PREPARA IL CUNEO
;-------------------------------------------------------------
          SEI         ;DISABILITA LA RICHIESTA D'INTERRUZIONE
          LDA $0314   ;RICEVE CONTENUTO DEL BYTE-BASSO DEL VETT.
          STA BBVETT  ;LO MEMORIZZA IN BBVETT
          LDA $0315   ;RICEVE CONTENUTO DEL BYTE-ALTO DEL VETT.
          STA BAVETT  ;LO MEMORIZZA IN BAVETT
          LDA #<CUNEO ;RICEVE L BYTE-BASSO DELL'INIZIO DEL CUNEO
          STA 0314    ;LO MEMORIZZA NEL BYTE-BASSO DEL VETT. IRQ
          LDA #>CUNEO ;RICEVE BYTE-ALTO DELL'INIZIO DEL CUNEO
          STA $0315   ;LO MEMORIZZA NEL BYTE-ALTO DEL VETT. IRQ
          CLI         ;RIABILITA LA RICHIESTA D'INTERRUZIONE
          RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
;RIMUOVE IL CUNEO
;-------------------------------------------------------------
          SEI         ;DISABILITA LA RICHIESTA D'INTERRUZIONE
          LDA BBVETT  ;RICEVE IL VALORE INIZIALE DI BBVETT
          STA $0314   ;REGISTRA NEL BYTE-BASSO DEL VETT. IRQ
          LDA BAVETT  ;RICEVE IL VALORE INIZIALE DI BAVETT
          STA $0315   ;LO REGISTRA NEL BYTE-ALTO DEL VETT. IRQ
          CLI         ;RIABILITA LA RICHIESTA D'INTERRUZIONE
          RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
;CICLO PRINCIPALE
;-------------------------------------------------------------
CUNEO     JSR REG     ;PASSA IL CONTROLLO A REG
          LDA TASTO   ;QUALE TASTO E' STATO PREMUTO ?
          CMP #$03    ;E' IL TASTO DI FUNZIONE #1 ?
          BNE CONT1   ;SALTA SE "NON VERO"
          JSR FLAG0   ;PASSA IL CONTROLLO A FLAG0
CONT1     LDA TASTO   ;QUALE TASTO E' STATO PREMUTO ?
          CMP #$04    ;E' IL TASTO DI FUNZIONE #7 ?
          BNE CONT2   ;SALTA SE "NON VERO"
          JSR FLAG1   ;PASSA IL CONTROLLO A FLAG1
CONT2     LDA TASTO   ;QUALE TASTO E' STATO PREMUTO ?
          CMP #$05    ;E' IL TASTO DI FUNZIONE #3 ?
          BNE CONT3   ;SALTA SE "NON VERO"
          JSR SOMMA   ;PASSA IL CONTROLLO A SOMMA
CONT3     LDA TASTO   ;QUALE TASTO E' STATO PREMUTO ?
          CMP #$06    ;E' IL TASTO DI FUNZIONE #5 ?
          BNE CONT4   ;SALTA SE "NON VERO"
          JSR SOTTR   ;PASSA IL CONTROLLO A SOTTR
CONT4     JSR REST    ;PASSA IL CONTROLLO A REST
          STX XCONT   ;MEMORIZZA IL VALORE DEL REG. X
          STY YCONT   ;MEMORIZZA IL VALORE DEL REG. Y
          JMP $EA31   ;TORNA AL PUNTO D'INTERRUZIONE
;-------------------------------------------------------------
;ROUTINE DI VERIFICA
;-------------------------------------------------------------
REST      LDA SUONA   ;RICEVE IL VALORE D'INNESCO
          BNE INIZIO  ;SE UGUALE A 1 SALTA
          RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
INIZIA    DEC RITARDO ;RIDUCE IL RITARDO
          BEQ AVVIA   ;SALTA SE 0
          RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
AVVIA     JSR CONT    ;PASSA IL CONTROLLO A CONT
          CPX #$10    ;FINE DEL CICLO ?
          BNE VERIF   ;SALTA SE NON 0
          JSR AZZERA  ;PASSA IL CONTROLLO A AZZERA
;-------------------------------------------------------------
VERIF     LDA RIGA1,Y ;RICEVE L'OFFSET DI RIGA1 DA Y
          BEQ SUCC1   ;SALTA SE 0
          JSR BATT1   ;PASSA IL CONTROLLO A BATT1
SUCC1     LDA RIGA2,Y ;RICEVE L'OFFSET DI RIGA2 DA Y
          BEQ SUCC2   ;SALTA SE 0
          JSR BATT2   ;PASSA IL CONTROLLO A BATT2
SUCC2     LDA RIGA3,Y ;RICEVE L'OFFSET DI RIGA3 DA Y
          BEQ SUCC3   ;SALTA SE 0
          JSR BATT3   ;PASSA IL CONTROLLO A BATT3
SUCC3     INY         ;INCREMENTA L'OFFSET
          INX         ;INCREMENTA IL CONTATORE DI CICLO
          RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
;SUBROUTINE
;-------------------------------------------------------------
FLAG0     LDA #$00    ;REGISTRA 0 -
          STA SUONO   ;IN SUONO
          RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
FLAG1     LDA #$01    ;REGISTRA 1 -
          STA SUONO   ;IN SUONO
AZZERA    LDX #$00    ;AZZERA IL REG. X
          LDY #$00    ;AZZERA IL REG. Y
          RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
CONT      LDA TEMPO   ;RICEVE IL VALORE DI TEMPO
          STA RITARDO ;LO REGISTRA IN RITARDO
          RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
SOMMA     LDA TEMPO   ;RICEVE IL VALORE DI TEMPO
          CMP #$FF    ;CONFRONTA IL RISULTATO CON 255
          BEQ CONT5   ;SALTA SE "VERO"
          INC TEMPO   ;INCREMENTA TEMPO
CONT5     RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
SOTTR     LDA TEMPO   ;RICEVE IL VALOERE DI TEMPO
          CMP #$01    ;CONFRONTA IL RISULTATO CON 1
          BEQ CONT6   ;SALTA SE "VERO"
          DEC TEMPO   ;DECREMENTA TEMPO
CONT6     RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
REG       LDX XCONT   ;MEMORIZZA IL VALORE DI XCONT NEL REG. X
          LDY YCONT   ;MEMORIZZA IL VALORE DI YCONT NEL REG. Y
          RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
;ROUTINE DI SUONO DELLA BATTERIA
;-------------------------------------------------------------
BATT1     LDA #$0E    ;PREPARA
          STA PERS1   ;
          LDA #$20    ;IL
          STA IMPULSO ;
          LDA #$4_    ;TAMBURO
          STA ONDA1   ;
          LDA #$03    ;BASSO
          STA BASSO   ;
          LDA #$41    ;E LO SUONA
          STA ONDA1   ;
          RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
BATT2     LDA #$07    ;PREPARA
          STA ATT2    ;
          LDA #$0C    ;IL
          STA PERS2   ;
          LDA #$80    ;TAMBURO
          STA ONDA2   ;
          LDA #$41    ;RULLANTE
          STA RULLANTE ;
          LDA #$81    ;E LO SUONA
          STA ONDA2   ;
          RTS         ;RESTITUISCE IL CONTROLLO
;-------------------------------------------------------------
BATT3     LDA #$02    ;PREPARA
          STA ATT3    ;
          LDA #$0D    ;LA
          STA PERS3   ;
          LDA #$12    ;CAMPANA
          STA ONDA3   ;
          LDA #$64    ;E
          STA CAMPANA ;
          LDA #$11    ;LA SUONA
          STA ONDA3   ;
          RTS         ;RESTITUISCE IL CONTROLLO
          .END
```

Sistemi di ricerca

Ricercare soluzioni di problemi è uno degli aspetti fondamentali dell'AI



STEVE CROSS

La corsa dei topi
Il diagramma mostra tre possibili percorsi attraverso un labirinto, associati a tre diverse strategie di ricerca: casuale, sistematica ed euristica. In questo esempio, il metodo euristico raggiunge per primo l'obiettivo, benché sia possibile concepire labirinti nei quali ciò non sia necessariamente vero. In pratica, comunque, per la soluzione di un labirinto si ricorre ad un metodo combinato tra quello euristico e quello "a esaurimento" (in cui vengono considerati tutti i movimenti)

Legenda:

- Metodo euristico
- Metodo sistematico
- Metodo casuale

Supponiamo di liberare tre topi in un labirinto che nasconde in un suo recesso una tazza di mangime. Uno dei ratti girella a caso per un paio di minuti poi si addormenta (quel furfante di sperimentatore gli ha versato qualche goccia di gin nell'acqua!). Il secondo topo è più metodico: rasentando le pareti del labirinto, egli svolta sempre a sinistra, tornando sui propri passi ogni volta che imbocca un vicolo cieco. La sua tenacia è premiata... ma dal ritrovamento di una tazza vuota, poiché il terzo topo è già a destinazione.

Quest'ultimo, infatti, ha un buon senso dell'olfatto e, ogni tanto durante la ricerca, si è soffermato per annusare l'aria e riconoscere la vicinanza del cibo. Ovviamente è possibile ideare un labirinto che confonda il topo che va "a naso" (o quello che segue il criterio metodico), ma va riconosciuto che la strategia di ricerca del terzo topo è la più intelligente.

Sia che siano nel Mar dei Caraibi a caccia di un tesoro o su di un treno a risolvere un cruciverba l'azione è sempre la stessa: si è alla ricerca di qualcosa. Lo studio sulle strategie di ricerca forma uno dei pricipali oggetti dell'AI e il comportamento dei tre topi ben rappresenta tre diversi tipi di strategia:

- Ricerca casuale (il "cammino dell'ubriaco")
- Ricerca "a esaurimento" (enumerazione sistematica)
- Ricerca euristica (esplorazione pilotata)

La terza è considerata la strategia più intelligente, poiché in genere porta alla soluzione del problema con il minimo sforzo. Tuttavia i metodi euristici si fondano su qualche sistema che consenta di sapere quando la ricerca si sta avvicinando alla soluzione (l'odore del cibo nel caso del topo). Senza questa conoscenza non si potrebbe essere "intelligenti" e bisognerebbe necessariamente seguire il criterio dell'enumerazione sistematica.

Trovare un percorso in un labirinto è un tipico problema di ricerca (sebbene tale metodo possa essere applicato con altrettanta efficacia anche nel caso dell'integrazione di un'espressione simbolica in matematica). Le risposte a due quesiti possono aiutare a valutare la qualità di un metodo di ricerca:

- Quanto tempo occorre a trovare la soluzione?
- Il percorso trovato è quello più economico?

Il metodo ottimale è quello che consente di trovare la soluzione ideale nel minor tempo possibile. In pratica, però, può rendersi ne-

QUESITI DI RICERCA

La maggior parte dei metodi di ricerca nel settore dell'Intelligenza Artificiale implica la raffigurazione dei possibili percorsi nel labirinto sotto forma di una struttura "ad albero". Il diagramma ad albero qui mostrato è generato prendendo come "radice" la casella di partenza del labirinto e ramificando da essa, prendendo in considerazione tutte le possibili uscite offerte da ciascun "nodo". Si noti che non sono ammessi movimenti diagonali tra caselle. Il diagramma di flusso sintetizza l'algoritmo impiegato per creare l'albero

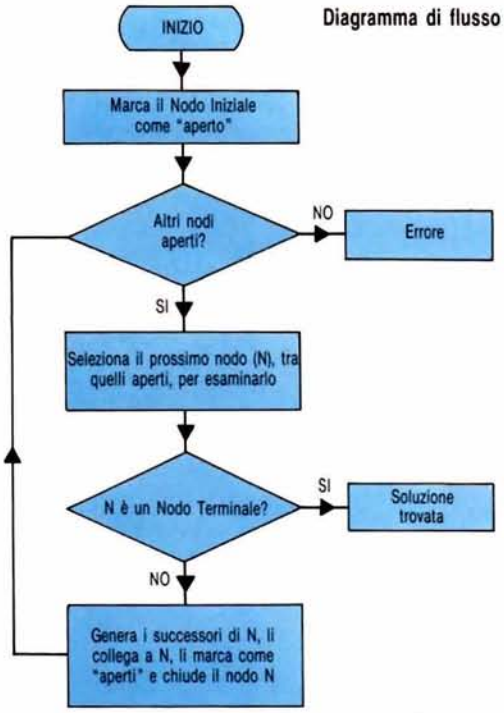
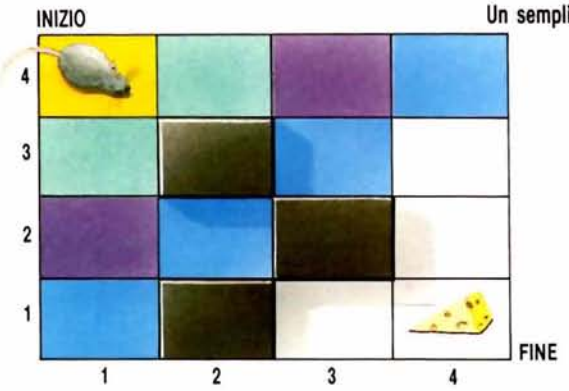
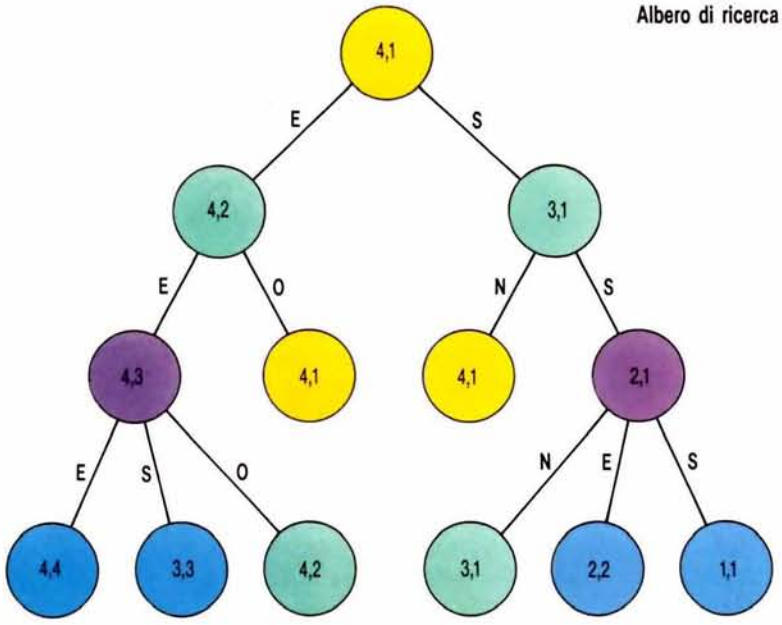


Diagramma di flusso



Un semplice labirinto



Albero di ricerca

KEVIN JONES

cessario un compromesso: una soluzione approssimata ma rapida oppure la soluzione migliore ma in tempi più lunghi.
Molte delle strategie di ricerca adottate nell'AI sono ideate partendo da un criterio comune, sul quale può avere predominanza l'una o l'altra tendenza. Il diagramma generale dell'algoritmo "di traversata" (mostrato a fianco) illustra una varietà di metodi di ricerca grazie ai quali poter scegliere un numero a seconda di come si desideri riempire una casella centrale (oppure quale nodo esaminare successivamente). I termini "aperto" e "chiuso" si riferiscono allo stato di ciascun nodo del percorso che va dalla partenza alla destinazione. I nodi da esaminare sono quelli aperti: i nodi chiusi sono quelli già esaminati. I nodi aperti, pertanto, si trovano come in una "lista d'attesa" e l'efficienza della ricerca dipende dall'ordine col quale viene esaminata tale lista.

Un albero di ricerca

La ricerca ramifica proprio come un albero. Nel semplice labirinto qui usato come esempio il topo si trova nella casella superiore sinistra e il cibo nella casella inferiore destra. In ogni casella il topo può scegliere tra quattro direzioni (nord, sud, est e ovest), sebbene alcune possano rappresentare ostacoli (i margini del labirinto, ad esempio). Se si riportano in un grafico le scelte concesse al topo, partendo dalla sua posizione iniziale, la struttura ad albero della ricerca risulta quanto mai evidente.
Dalla casella iniziale (4,1) il ratto può muoversi soltanto a est (4,2) o a sud (3,1). Da (4,2) può muoversi a est oppure a ovest e da (3,1) a nord oppure a sud. Se il topo si muovesse ad est e poi a ovest (oppure a sud e poi a nord) egli ritornerebbe al punto di partenza, compiendo un movimento insensato. Tuttavia ciò dimostra che uno stesso nodo può apparire più volte nell'albero, a seconda del percorso impiegato per accedere ad esso.
Un metodo di esplorazione sistematica dell'albero è quello chiamato "per ampiezza", in cui i nodi vengono esaminati in ordine di prossimità rispetto a quello di partenza (o "radice"). Pertanto tutte le sequenze di N movimenti (livello N dell'albero) vengono considerate prima di quelle che hanno N+1 movimenti. Il metodo conduce alla soluzione che richiede il minimo numero di passi e, purché ogni passo implichi un "costo" identico, anche a quella più economica. Tuttavia, lo scotto che si paga con questo criterio è una lunga attesa: se la complessità dell'albero cresce, aumenta anche (in modo esponenziale) il tempo occorrente per trovare la soluzione. La ricerca "per ampiezza" può essere migliorata introducendo un frammento di informazione euristica circa la distanza dell'obiettivo da raggiungere. Per valutare la distanza si può quindi applicare il cosiddetto "Metodo di Manhattan". Sull'isola di Manhattan, nello stato di New York, la maggior

parte delle strade si interseca ad angolo retto con le altre cosicché, per arrivare da A a B, occorre spostarsi, ad esempio, di tanti isolati a nord e di tanti isolati a est. Per analogia, un topo "intelligente" può calcolare quanto dista, in caselle, l'obiettivo.
Se si sa quando la ricerca si avvicina alla soluzione, si possono prendere delle scorciatoie. Il terzo topo dell'esempio iniziale, in sostanza, segue una semplice regola: avvicinarsi sempre più all'obiettivo. Questa strategia, detta anche "a scalata" paragonandola alla ricerca della sommità di una collina avvolta nella nebbia (la cima si può trovare anche alla cieca, se si procede sempre in salita), può risultare più rapida della ricerca "per ampiezza", ma non è detto che il percorso risultante sia quello ottimale. Inoltre, può capitare di bloccarsi su di un picco "locale" (la ricerca è indirizzata a una sommità qualsiasi).

Un metodo migliore è quello che risulta da un compromesso tra le due teorie appena esposte. L'algoritmo impiegato, che prende il nome di A-Star (A*), seleziona il prossimo nodo da esaminare in base all'espressione DE + DP, dove DE è la stima euristica della distanza rimanente e DP è la distanza percorsa: l'efficacia della ricerca dipende dlla precisione della stima di DE. In ogni caso, conviene che DE indichi una distanza per difetto piuttosto che una per eccesso.
Nel programma dimostrativo qui presentato vengono illustrati tutti e tre i metodi esposti, apportando solo piccole modifiche. L'unica differenza di rilievo, in effetti, è il criterio di scelta del "prossimo" nodo da esaminare:

Ricerca "per ampiezza" — sceglie DP minima
Ricerca "a scalata" — sceglie DE minima
Algoritmo A* — sceglie DE+DP minima

PROGRAMMA DIMOSTRATIVO PER IL LABIRINTO

```
1000 REM**************************************
1010 REM**  Listato 2.1:                    **
1020 REM**  SOLUZIONE DI LABIRINTI          **
1030 REM**************************************
1040 MODE7
1050 MH=17: MW=25: REM Profondita' e Larghezza labirinto
1060 SI=256: REM limiti dell'albero
1070 WA=1: RR=2: FO=3: DN=4: BL=5
1080 W1=1: REM "peso" per DP
1090 W2=2: REM "peso" per DE
1100 DIM M(MH+1,MW+1): REM matrice del labirinto
1110 DIM C$(5): REM caratteri per il labirinto
1120 DIM P(SI),S(SI),N(SI),H(SI)
1130 REM Percorso, Passi,Nodo,Distanza euristica
1140
1150 REM -- Topo nel Labirinto:
1160 GOSUB 1360: REM crea il labirinto
1170 NC=0: REM n. di nodi esaminati
1180 K=0: REM contatore
1190 GOSUB 1660: REM "ripulisce" l'albero
1200 N(1)=2*MW+2: REM 1mo nodo aperto
1210 S(1)=0: H(1)=FR-1 + (FC-1)
1220 P(1)=0: REM n. del precedessore
1230 REM -- Ciclo principale:
1240 REM *** CICLO PRINCIPALE ***
1250 GOSUB 1770: REM  selez. nodo successivo S
1260 NC=NC+1
1270 PRINT TAB(0,22);NC,SR,SC,H(S);"
1280 GOSUB 1880: REM genera successori
1290 IF (FR<>SR OR FC<>SC) AND NC<300 THEN 1240
1300 PRINT TAB(0,22);"Ricerca conclusa!"
1310 IF FR=SR AND FC=SC THEN GOSUB 2290:
     REM indietreggia
1320 IF NC>300 THEN PRINT "Tentative fallito!"
1330 PRINT NC;" nodi esaminati."
1340 END
1350 :
1360 REM -- Routine per la creazione del labirinto:
1370 FOR P=1 TO MH+1
1380 FOR R=1 TO MW+1
1390 IF RND(1) < 0.28 THEN M(P,R)=WA ELSE M(P,R)=BL
1400 IF P=3 OR R=3  THEN M(P,R)=BL
1410 IF P=1 OR R=1 THEN M(P,R)=WA
1420 IF P>MH OR R>MW THEN M(P,R)=WA
1430 NEXT:NEXT
1440 FR=2+INT(RND(1)*MH-1)): REM riga del cibo
1450 FC=4+INT(RND(1)*MW-3: REM colonna del cibo
1460 M(FR,FC)=FO
1470 M(2,2)=RR: REM topo robot
1480 C$(BL)=" "
1490 C$(WA)=CHR$255: REM blocco
1500 C$(DN)=" "
1510 C$(RR)="R"
1520 C$(FO)="f"
1530 GOSUB 1560: REM visualizza labirinto
1540 RETURN
1550 :
1560 REM -- Routine per visualizzare il labirinto
1570 CLS
1580 FOR R=1 TO MH+1
1590 FOR C=1 TO MW+1
1600 PRINT TAB(C,R); C$(M(R,C));
1610 NEXT
1620 PRINT
1630 NEXT
1640 RETURN
1650 :
1660 REM -- Routine che "ripulisce" l'albero
1670 DD=9999: REM morto
1680 FOR Q=1 TO SI
1690 P(Q)=0
1700 S(Q)=DD
1710 N(Q)=0
1720 H(Q)=DD
1730 NEXT
1740 NN=2: REM prossimo nodo libero
1750 RETURN
1760 :
1770 REM -- Sceglie il miglior nodo S:
1780 S=1: BN=DD
1790 FOR I=1 TO SI
1800 V=S(I)*W1 + ABS(H(I))*W2
1810 IF V<BN AND H(I)>=0 THEN S=I: BN=V
1820 NEXT
1830 IF S=1 THEN PRINT TAB(0,20);"Sto esplorando..."
1840 SR=INT(N(S)/MV)
1850 SC=N(S)-MW*SR
1860 RETURN
1870 :
1890 REM -- Routine che genera i successori:
1900 REM -- Nord:
1910 Y=SR-1: X=SC
1920 IF Y>1 THEN GOSUB 2090
1930 REM -- Est:
1940 Y=SR: X=SC+1
1950 IF X<=MW THEN GOSUB 2090
1960 REM -- Sud:
1970 Y=SR+1: X=SC
1980 IF Y<=MH THEN GOSUB 2090
1990 REM -- Ovest:
2000 Y=SR: X=SC-1
2010 IF X>1 THEN GOSUB 2090
2020 REM -- "chiude" il nodo S:
2030 H(S) = -H(S)
2040 IF H(S)>0 THEN PRINT "Ugh!"
2050 PRINT TAB(SC,SR);" ";
2060 M(SR,SC)=DN
2070 REM pulisce una cella di schermo
2080 :
2090 REM -- Routine per aprire 1 nodo:
2100 IF M(Y,X)=DN THEN RETURN
2110 IF M(Y,X)=WA THEN RETURN
2120 REM -- trova 1ma locazione libera:
2130 NX=0
2140 REM ** CICLO DI RICERCA LOCAZIONE **
2150 IF S(NN)<>DD THEN NX=NX+1: NN=NN+1
2160 IF NN>SI THEN NN=1
2170 IF NX>SI THEN PRINT "Troppo pieno!": STOP
2180 IF S(NN)<>DD THEN 2140
2190 REM -- Adesso apre il nodo:
2200 XY=X + Y*MW
2210 N(NN)=XY
2220 P(NN)=S
2230 S(NN)=X(S)+1
2240 H(NN)=ABS(Y-FR) + ABS(X-FC)
2250 PRINT TAB(X,Y);"+";
2260 REM visualizza nodo sullo schermo
2270 RETURN
2280 :
2290 REM -- routine di indietreggiamento:
2300 ST=S(S)
2310 FOR Q=1 TO 10000: NEXT Q: GOSUB 1560
2320 PRINT TAB(FC,FR); C$(FO);
2330 REM ** VISUALIZZA PERCORSO **
2340 S=P(S): REM nodo "genitore"
2350 XY=N(S): REM coordinate
2360 Y=INT(XY/MW)
2370 X=XY-Y*MW
2380 M(Y,X)=RR: REM impronte del topo!
2390 PRINT TAB(X,Y);"*";
2400 IF S>0 THEN 2330
2410 PRINT TAB(2,2);C$(RR)
2420 PRINT TAB(0,22);"Il percorso contiene ";ST;
     " passi."
2430 PRINT NC;" nodi chiusi."
2440 RETURN
```

Linee BASIC per un labirinto

Il labirinto è contenuto in una matrice bidimensionale M(,), mentre le strutture dati necessarie per la ricerca nell'albero sono nelle matrici P( ), S( ), N( ) e H( ). Il programma combina un fattore di "costo attuale" e un fattore di "costo per raggiungere l'obiettivo". A ciascuno di questi fattori può essere attribuito un "peso", modificando il valore di W1 e di W2 (linee 1080 e 1090). Con i valori qui specificati, il programma segue una strategia di ricerca "a scalata", ma consente di eseguire altre sperimentazioni. Il criterio euristico adottato dal programma è basato sul metodo di misura delle distanze detto di "Manhattan" che, nel caso specifico, risulta molto preciso. Pertanto, attribuendo un "peso" maggiore al criterio euristico (facendo diventare W2 > W1), la ricerca diventa più rapida

Basic & Basic

Il programma a fianco è scritto per il Micro BBC. Per il Commodore 64 e per lo Spectrum apportare le seguenti modifiche.

Commodore 64

Sostituire le espressioni PRINT TAB(X,Y);"MESSAGGIO" con:

```
P=X:Q=Y:GOSUB 3000:
  PRINT "MESSAGGIO"
```

e inserire:

```
3000 REM * ROUTINE TAB *
3010 PRINT CHR$(19);
3020 PRINT LEFT$(DW$,Q);
     TAB(P);
3030 RETURN

1040 FOR I=1 TO 25:
     DW$=DW$+CHR$(17):
     NEXT I

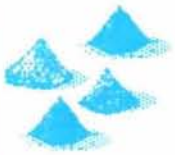
1490 C$(WA)=CHR$(102)
1570 PRINT CHR$(147)
```

Spectrum:

Sostituire le espressioni TAB(X,Y); con AT X,Y;

```
1490 LET C$(WA)=
     CHR$(143)
```

Cancellare la linea 1040

	(1)	(2)	(3)	
T$()	 Perle	 Manufatti	 Spezie	Descrizioni dei prodotti offerti dal capo
V1()	2	2	1	Valore, in pezzi d'oro, al momento della partenza
V2()	?	?	?	Valore di mercato quando la nave rientra in porto
Q(,1)	0.5	0.5	1	 1 Balla di sale
	5	5	10	 1 Balla di stoffe
	3	3	6	Tassi di scambio per i prodotti commerciati  1 Monile
	2	2	4	 1 Coltello

Approdo al Nuovo Mondo

Se si riescono ad evitare scontri con gli indigeni può iniziare il commercio

Nella fase finale del gioco, il viaggio in sé si conclude approdando con successo sulle rive del Nuovo Mondo e instaurando un rapporto commerciale grazie ai materiali di scambio acquistati alla partenza. I pericoli, tuttavia, non sono ancora terminati: nulla si sa del comportamento degli indigeni, che possono essere amichevoli oppure ostili. Occorre quindi evitare azioni che possano causare antagonismo tra loro e l'equipaggio della nave.
Quando nel programma termina il ciclo principale della traversata, la linea 891 chiama la subroutine alla linea 10000, che si occupa dell'arrivo nel Nuovo Mondo. Mentre la nave si accinge ad approdare, diversi gruppi di indigeni armati si avvicinano in esplorazione. Al giocatore spetta decidere se evitare di aprire il fuoco, rischiando così un assalto da parte degli esploratori, oppure iniziare a sparare e causare molto probabilmente uno scontro.

La linea 10015 esamina il secondo elemento della matrice delle scorte, OA(2), che indica quanti fucili ci sono a bordo. Se ne esistono, occorre decidere se usarli o no. La linea 10022 attende la risposta e, se questa è affermativa, molti indigeni rimarranno uccisi nello scontro. Una simile scelta, però, può implicare che gli indigeni superstiti, durante la notte, attacchino la nave incendiandola (il che equivale, ahimè, alla fine del gioco!).
È ovvio che un'aggressione ingiustificata non porta a risultati positivi e la linea 10044 impedisce che il gioco prosegua dopo una simile decisione.
Se si decide di non sparare, non c'è alcun motivo per una rappresaglia, pertanto la linea 10026 esegue una diramazione alla linea 10050. In questa sezione del programma, i nativi vengono issati a bordo amichevolmente ed offrono il loro benvenuto all'equipaggio, che viene poi trasportato a terra per

incontrarsi con il capo. Dopo abbondanti festeggiamenti e libagioni, un nuovo giorno apre le trattative per il commercio (di ciò si occupa un modulo a parte, trattato in seguito). Prima di impostare le trattative, però, vanno aggiustate alcune cose.

Le matrici per il commercio

Un fatto di cui tener conto è che il valore delle perle, delle spezie e dei manufatti, determinato prima della partenza, è nel frattempo mutato.

Per facilitare il commercio occorre creare nuove matrici: la linea 60 DIMensiona la matrice T$(), che contiene le descrizioni dei tre prodotti offerti dal capo indigeno: perle, manufatti e spezie. La linea 61 dimensiona una matrice, V1(), che contiene i valori di mercato di questi prodotti al momento della partenza della nave per il viaggio.

Il valore iniziale delle perle e dei manufatti era di due pezzi d'oro ciascuno e quello delle spezie era di un pezzo d'oro al grammo. La linea 62 dimensiona la matrice V2(), che contiene i prezzi dei prodotti quando la nave rientra al porto di partenza. Poiché i valori di mercato fluttuano, su di essi influisce un elemento casuale. Il prezzo in V2(1), relativo alle perle, viene fissato in 2 1/2 pezzi d'oro dall'espressione nella linea 62. Altre espressioni simili, nella medesima linea, alterano il prezzo degli altri prodotti.

Il commercio dei prodotti, comunque, non avviene in base a pezzi d'oro, ma attraverso lo scambio di merci. Ad esempio il capo offre al capitano una certa quantità di perle, di manufatti e di spezie in cambio di un "lotto" di merci offerte dal capitano. Per semplificare le cose, il programma suppone che gli indigeni abbiano risorse illimitate.

Una matrice bidimensionale, contenente i "tassi di cambio" per ciascuna merce, viene dimensionata e riempita di valori nelle linee da 64 a 68. Il primo indice corrisponde ai quattro prodotti offerti dal capitano: sale, stoffa, coltelli e monili, mentre il secondo si riferisce ai prodotti offerti dagli indigeni: perle, manufatti e spezie, i cui tassi di cambio sono fissati da alcune linee di programma.

La linea 64 fissa il tasso di scambio per i sacchi di sale. Le perle sono il primo elemento corrispondente al secondo indice e ad EQ(1,1) viene assegnato un valore che indica il rapporto tra perle e sacchi di sale. Fissando EQ(1,1) a 0.5 significa che un sacco di sale equivale a mezza perla. Analogamente, EQ(1,2) fissa il rapporto tra il valore del sale e quello dei manufatti.

Quando EQ(1,2) vale 0.5 significa che occorrono due sacchi di sale per un manufatto. EQ(1,3) = 1 significa che un grammo di spezie vale un sacco di sale.

I valori di scambio relativi alle balle di stoffe sono fissati dalla linea 66. Una balla equivale a 5 perle, 5 manufatti o 10 grammi di spezie. La linea 67 si occupa del tasso di scambio per i coltelli: ognuno di questi vale rispettivamente tre perle, tre manufatti o sei grammi di spezie. I tassi di scambio per i monili, ultimo elemento della matrice, vengono fissati alla linea 67: due perle, due manufatti o quattro grammi di spezie.

Alla linea 69 viene dimensionata una matrice, AO(3), per conservare la quantità di perle, manufatti e spezie acquistate durante le trattative. Questa matrice non va confusa con OA(), che contiene le quantità di acquisti fatti alla partenza.

MODULO UNDICI: APPRODO

Dimensionamento delle matrici occorrenti

```
60 DIMT$(3): T$(1)="PERLE": T$(2)="MANUFATTI":
   T$(3)="SPEZIE"
61 DIMV1(3): V1(1)=2: V1(2)=2: V1(3)=1
62 DIMV2(3): V2(1)=2+(INT(RND(1)*1)/2):
   V2(2)=2+(INT(RND(1)*3)-1)
63 V2(3)=2+(INT(RND(1)*1)/2)
64 DIMEQ(4,3)
65 EQ(1,1)=0.5: EQ(1,2)=0.5: EQ(1,3)=1
66 EQ(2,1)=5: EQ(2,2)=5: EQ(2,3)=10
67 EQ(3,1)=3: EQ(3,2)=3: EQ(3,3)=6
68 EQ(4,1)=2: EQ(4,2)=2: EQ(4,3)=4
69 DIMAO(3)
```

Aggiunte al programma principale

```
890 REM APPRODO AL NUOVO MONDO
891 GOSUB 10000
```

Subroutine dell'approdo

```
10000 REM APPRODO AL NUOVO MONDO
10001 PRINT CHR$(147): GOSUB 9200
10005 S$="SEI GIUNTO NEL NUOVO MONDO•": GOSUB 9100
10006 PRINT: GOSUB 9200
10007 S$="MENTRE LA NAVE SI AVVICINA•": GOSUB 9100
10008 S$=" VENGONO INCONTRO ALCUNI•":GOSUB 9100
10009 S$=" INDIGENI •":GOSUB 9100
10010 PRINT: GOSUB 9200
10015 IF OA(2)=0 THEN 10050
10017 S$="SONO ARMATI E BELLICOSI!!": GOSUB 9100
10018 PRINT: GOSUB 9200
10020 S$="APRI IL FUOCO (S/N)?•": GOSUB 9100
10022 INPUT I$: I$=LEFT$(I$,1)
10024 IF I$<>"N" AND I$<>"S" THEN 10022
10026 IF I$="N" THEN 10050
10028 PRINT: GOSUB 9200
10030 S$="MOLTI INDIGENI RIMANGONO UCCISI•":GOSUB 9100
10032 S$=" MA DURANTE LA NOTTE•": GOSUB 9100
10034 S$=" NE ARRIVANO ALTRI•": GOSUB 9100
10036 S$=" CHE BRUCIANO LA TUA NAVE!!!!•": GOSUB 9100
10038 PRINT: PRINT: GOSUB 9200
10040 S$="          GAME OVER•": GOSUB 9100
10042 END
10044 GOTO 10042
10050 S$="TI CONDUCONO A CONOSCERE IL•": GOSUB 9100
10052 S$=" LORO CAPO, CHE HA GIA'": GOSUB 9100
10053 S$=" COMMERCIATO CON EUROPEI.•":GOSUB 9100
10054 S$=" E' MOLTO AMICHEVOLE.•": GOSUB 9100
10056 GOSUB 9200
10058 S$="L'EQUIPAGGIO VIENE SFAMATO•":GOSUB 9100
10059 S$="E SI RIPOSA.•":GOSUB 9100
10060 PRINT: GOSUB 9200
10062 S$="IL GIORNO DOPO INIZIANO•":GOSUB 9100
10063 S$="LE TRATTATIVE.•":GOSUB 9100
10064 PRINT: GOSUB 9200
10066 S$=K$: GOSUB 9100
10068 GET I$: IF I$="" THEN 10068
10069 RETURN
```

Basic & Basic

Spectrum:

Sostituire V1( ) con B( ), V2( ) con D( ), EQ(,) con Q(,) e AO( ) con E( ) in tutto il programma e apportare le seguenti modifiche:

```
   60 DIM T$(3,9)
10001 CLS:GO SUB 9200
10022 INPUT I$:LET I$ = I$(TO 1)
10068 LET I$ = INKEY$:IFI$ = "" THEN GOTO 10068
```

Micro BBC:

Apportare le seguenti modifiche

```
10001 CLS: GOSUB 9200
10068 I$ = GET$
```

IAN McKINNELL sul Macintosh

GOTO END

Nella parte conclusiva sul PASCAL gli ultimi consigli e suggerimenti

L'ideatore del linguaggio PASCAL, Niklaus Wirth, ha intitolato uno dei suoi libri *Strutture dati + Algoritmi = Programmi*, evidenziando quanto sia importante un'esatta descrizione dei dati nella formulazione di algoritmi che operano su di essi. Se ad esempio, si può usare una matrice di char per rappresentare una stringa, le funzioni e le procedure per elaborare tale elemento (per trovarne la lunghezza, per concatenarla ad altre stringhe ecc.) sono notevolmente diverse da quelle necessarie per elaborare una lista ad anello.
Si prenda, ad esempio, il semplice compito di trovare la lunghezza di una stringa, quando essa è inizialmente sconosciuta. Si rammenti che gli elementi inutilizzati della matrice usata per contenere la stringa vengono riempiti con il carattere ASCII NUL, ossia chr(0) o che la lista dinamica termina con un puntatore il cui valore è NIL. La versione con matrice di caratteri è abbastanza semplice:

```
FUNCTION Lungh (S :string) : Cardinale;
   VAR
      N            : 0..LunghStringa;
      trovato      : boolean;
   BEGIN
      N := 0;
      trovato := false;
      REPEAT
         N := N + 1;
         trovato := S[N] = chr(0)
      UNTIL trovato OR (N = LunghStringa);
      IF trovato
         THEN
            Lungh := N — 1
         ELSE
            Lungh := LunghStringa
   END; {Lungh}
```

È facile a questo punto commettere errori o fare confusione; ad esempio: perché usare la variabile locale trovato? Perché occorre assegnare a Lungh il valore N — 1? Questi sono piccoli dubbi, ma se l'algoritmo fosse più comples-

PROGRAMMA RIORDINATORE DI ALBERI

Nel programma TreeSorter qui presentato vengono adoperate alcune procedure già viste nel programma ListeCirc. I riferimenti a queste si trovano nel listato. I dati, che consistono di nomi, somme e qualsiasi altro dato si desideri aggiungere, vengono inseriti da tastiera e collocati dal programma, in ordine alfabetico ascendente, in una struttura ad albero binario.
Ogni nodo dell'albero possiede due puntatori, per collegare l'elemento "precedente" e quello "successivo". L'accesso trasversale all'albero è conseguito confrontando ogni nuova immissione con il nome contenuto in ciascun nodo e scegliendo il ramo appropriato (RamoInf o RamoSup) nei due campi dei puntatori. Quando si trova un nodo vuoto (il relativo puntatore vale NIL), viene inserito il nuovo elemento. La scrittura dei dati nel file viene quindi fatta semplicemente e naturalmente per mezzo di una procedura ricursiva.
L'esempio qui riportato può essere adoperato per creare un programma di gestione per archivi di concezione molto elegante. Occorre però un avvertimento: se un file ordinato alfabeticamente viene trasferito in un albero binario, tutti gli inserimenti avvengono sullo stesso ramo, cosicché si ha come risultato non più un albero binario ma una lista ad anello singolo!
Per le applicazioni più semplici, la lista circolare è forse la più versatile: in essa non esistono puntatori con valore NIL (quindi si può omettere parte del codice che esegue il confronto su ogni nodo). Un albero binario, invece, ha un numero di NIL pari a (Numero di Nodi + 1).

```
PROGRAM    TreeSorter (input, output, FileDati);

CONST
        NomeFile      = 'Treedata';
        LunghStringa  = 25;

TYPE
   Cardinale        = 0..MaxInt;
   DimStringa  = 1..LunghStringa;
   stringa          = PACKED ARRAY [DimStringa]
                          OF char;
   cosa             = RECORD
                          Nome    : stringa;
                          (altri campi..)
                          ammontare    : Cardinale
                       END; (cosa)

   TipoFile        = FILE OF cosa;
   albero          = ^ fronda;
                     (** riferimento in avanti a :)
   fronda          = RECORD
                        elem          : cosa;
                        RamoInf,
                        RamoSup   : albero
                      END; (fronda)

VAR
   dato            : cosa;
   FileDati        : TipoFile;
   tronco          : albero;

INCLUDE 'Utils.src' (** file sorgente contenente:
                 SaltaSpazi, LeggiScheda e LeggiRiga
                         - vedere programma ListaCirc **)
      (1111111111111111111111111111111111111)

PROCEDURE  LeggiAmmontare  (VAR cifra  :  Cardinale);
       (** Verifica validita' di un valore Cardinale **)
   VAR
      OK        : boolean;
   BEGIN
     REPEAT
       REPEAT
         write ('Ammontare?' : 20);

         IF EoL (input) THEN
           ReadLn (input);

         SaltaSpazi (input)
         (** nuovo prompt se fine di linea - EoLn -  **)
       UNTIL NOT EoLn (input);

       LeggiScheda (input, cifra, OK);

        IF NOT OK THEN
         WriteLn ('--- ERRORE --- ' : 20,
                        're-inserire')
     UNTIL OK    (** prosegue finche' numero e' valido **)
```

so le cose potrebbero complicarsi. Usare segnalatori o "flag" (tipo booleano) è uno dei trucchi usati dai programmatori, ma talvolta il loro impiego è dovuto più alle necessità del computer che non a quelle strettamente pertinenti all'algoritmo ideato. Va usata la variabile locale N e non Lungh, che è un identificatore di funzione, non una variabile. Porre un identificatore di funzione nella parte destra di un'espressione del tipo:

```
Lungh := Lungh + 1
```

rappresenterebbe un tentativo di invocare una chiamata ricursiva alla funzione Lungh.
Si confronti ciò con la funzione Lungh occorrente per stringhe rappresentate come liste ad anello. Nella rappresentazione dinamica occorre ricordare che la definizione di string come TYPE è notevolmente diversa, permettendo la creazione di stringhe di qualsiasi lunghezza. Simili descrizioni di dati vengono spesso definite in modo ricursivo.

La ricursione

Molti esempi banali spesso adoperati per illustrare la ricursione potrebbero essere espressi altrettanto bene impiegando algoritmi iterativi, ma guardiamo alla funzione Lungh per il tipo di stringa ricursiva:

```
FUNCTION Lungh (S :string)     : Cardinale;
   BEGIN
      IF S = NIL
         THEN
            Lungh := 0
         ELSE
            Lungh := succ(Lungh(S^.prox))
   END;        Lungh
```

La testa della lista è S e l'espressione Sprox seleziona il campo del puntatore per il prossimo record nella lista. In alternativa, si può pensare che la lista inizi con il "prossimo" record (ButFirst). Quando non si incontra un NIL, viene richiamato il calcolo della lunghezza di ButFirst, incrementandolo con la funzione succ.
Esistono molti altri problemi la cui soluzione è possibile solo affidandosi alla ricursione. In effetti, l'esempio appena visto può essere riscritto non usando la ricursione, ma in tal caso occorre introdurre una variabile locale, che funge da contatore (come nella versione a matrice di caratteri):

```
    END;   (LeggiAmmontare)
          (11111111111111111111111111111111111111111)

PROCEDURE      Crescita     (VAR   foglia    : albero;
     (** innesto sull'albero **)  dato    : cosa);
  BEGIN
    new (foglia);

    WITH foglia ^ DO
      BEGIN
        elem := dato;
        RamoInf := NIL;  (** Nessun altro dato, per ora,)
        RamoSup := NIL; (oltre questa fronda **)
      END

  END;  (Crescita)
        (11111111111111111111111111111111111111111)

PROCEDURE      Arrampicata       (germoglio    : albero;
  (** trova dove eseguire inserimento **) dato     : cosa);
  VAR
        stelo    : albero;
        maggiore  : boolean;
  BEGIN
    WHILE  germoglio <> NIL DO
      BEGIN      (** trova una fronda vuota **)
        stelo  :=  germoglio;
        maggiore := dato . Nome >
                    stelo ^ . elem . Nome;
        IF maggiore
           THEN (** si muove verso l'alto **)
             germoglio := germoglio ^ . RamoSup
           ELSE (** si muove verso il basso **)
             germoglio := germoglio ^ . RamoInf
      END;

    Crescita  (germoglio, dato);

    IF maggiore
      THEN        (** inserisce sopra **)
        stelo ^ . RamoSup := germoglio
      ELSE        (** inserisce sotto **)
        stelo ^ . RamoInf := germoglio

  END;  (Arrampicata)
        (11111111111111111111111111111111111111111)

PROCEDURE       Visualizza (VAR       F      : TipoFile;
  (** scrive dati in un file **) radice  : albero);
  BEGIN
    IF radice <> NIL THEN
      WITH radice ^ DO  (** ricursione :)
        BEGIN   (** scrive in ordine :)
          Visualizza (F, RamoInf); (** inferiore per primo,)
          write (F, elem); (la radice stessa,)
          Visualizza (F, RamoSup) (maggiore per ultimo **)
        END
```

```
     END;  (Visualizza)
          (11111111111111111111111111111111111111111)

PROCEDURE          Recupera    (VAR radice    : albero);
  (** recupera memoria allocata per riutilizzo **)
  BEGIN
    IF  radice <> NIL THEN (** ricursione **)
      BEGIN        (** recupera per primi i rami -)
        Recupera  (radice ^ . RamoInf);
        Recupera  (radice ^ . RamoSup);
        dispose  (radice) ( - poi la radice **)
      END

  END;  (Recupera)
        (11111111111111111111111111111111111111111)

BEGIN    (TreeSorter - Programma Pincipale)

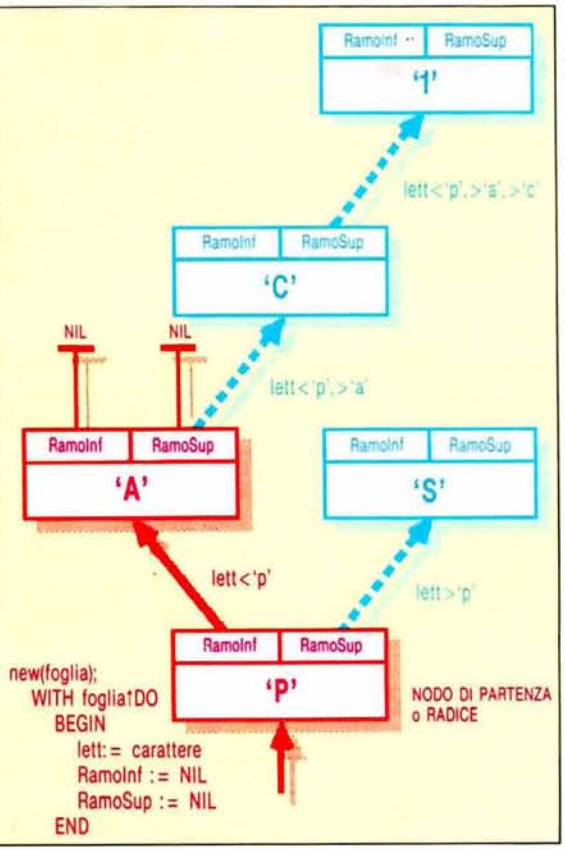
  assign (FileDati, NomeFile);
  page (output);
  WriteLn ('== Sort di un albero ==' : 25);
  WriteLn;
  WriteLn ('Immetti un dato (cognome per primo)' );
  write  ('Nome?');
  LeggiRiga (dato . Nome);
  LeggiAmmontare (dato . ammontare);
  Crescita (tronco, dato); (* pianta il tronco dell'albero *)
  write ('Nome? ');
  LeggiRiga (dato.Nome);

  WHILE dato . Nome [1] <> chr (0) DO
    BEGIN
        LeggiAmmontare (dato . ammontare );
        Arrampicata (tronco, dato);
        WriteLn ('RETURN per finire' : 40);
        write ('Nome?');
        LeggiRiga (dato . Nome)
    END;
    (** altre elaborazioni, poi scrive sul file : )
  WriteLn ('Sto scrivendo il dato in : ', NomeFile);
  rewrite (FileDati);
  Visualizza (FileDati, tronco);
  WriteLn;
  WriteLn ('Ammontare' : 8, 'Nome' : LunghStringa);
  WriteLn;
  reset (FileDati); (** legge il file,)
        (e visualizza i dati in ordine :)
  WHILE NOT EoF (FileDati) DO
   BEGIN        (** legge ciascun record **)
        read (FileDati, dato);
          (** emette i campi sullo schermo : )
        WITH dato DO
          WriteLn (ammontare : 8, ':', Nome)
    END;

  Recupera (tronco);  (** recupero memoria)
    (per ulteriori elaborazioni ecc. **)
END.
```

Nasce una nuova foglia

Il programma "TreeSorter" adopera una struttura ad albero binario per conservare dati ordinati in base a una chiave (quella alfabetica). Gli elementi vengono inseriti nel ramo inferiore o superiore a seconda della posizione alfabetica del contenuto del campo riservato al nome. La procedura "Crescita" crea nuovi nodi, mentre la procedura "Risali" trova il giusto percorso per arrivare ad un ramo NIL.
Il diagramma illustra una semplice struttura ad albero binario, in cui ogni nodo contiene soltanto un carattere (lett), e mostra come viene conservata una stringa di caratteri minuscoli (PASCAL). La parte in rosso indica una fase nella crescita dell'albero, di cui si occupa la procedura Crescita



LIZ DIXON sul Macintosh

```
FUNCTION Lungh (S :string)     : Cardinale;
  VAR
     N     : Cardinale;
  BEGIN
     N := 0;
     WHILE S <> NIL DO
        BEGIN
           N := N+1;
           S := S^.prox
        END;
     Lungh := N
  END; {Lungh}
```

Anche con questo algoritmo iterativo, la chiarezza e concisione sono apparenti e date solo dalla natura ricursiva della struttura dati. Molti compilatori PASCAL prevedono l'uso di "direttive": istruzioni dirette al compilatore e non vere e proprie istruzioni o comandi. L'unica direttiva prevista dallo Standard ISO, per adesso è Forward. Se due procedure o funzioni devono chiamarsi a vicenda, vengono dette "mutualmente ricursive". Ciò accade di rado, ma pone un problema: non si può usare nessun oggetto PASCAL, finché esso non sia dichiarato o definito.
La soluzione consiste nel dichiarare soltanto la testata di un sottoprogramma, sostituendo al suo blocco la direttiva FORWARD. Una volta completata la definizione dell'altro modulo, la testata viene riportata in forma abbreviata (omettendo la lista dei parametri) e il blocco viene definito in tale fase. Esistono altre direttive che influiscono sul funzionamento del compilatore, ma il loro uso andrebbe limitato al minimo se si desidera mantenere una certa trasportabilità del programma. Infatti, a parte lo speciale simbolo (in genere $) che appare all'inizio dei commenti, non è detto che le altre direttive vengano riconosciute da un altro compilatore.

PASCAL & PASCAL

Un paio di versioni non standard (tra le quali la HiSoft) richiedono una sintassi diversa per la dichiarazione di puntatori "in avanti" (Forward). In caso di dubbio consultare il manuale della propria versione. Comunque le varianti di questo genere, nei compilatori che seguono lo standard ISO, sono poche.
Un'altra differenza che si rileva nelle versioni UCSD, TCL/RML/Oxford e HiSoft è la mancanza di una procedura predefinita dispose. Al posto di questa vengono offerte le procedure non standard mark e release.
Esiste, infine, un ultimo tipo di descrizione complessa per i dati, in PASCAL, alla quale non si è finora accennato: la "variante". Quando è stato necessario conservare elementi di diverso tipo, si è usato un record con campi definiti appropriatamente. Ma supponendo che uno dei campi (la descrizione, ad esempio) o anche l'intero record debba essere più elastico, come comportarsi? Un esempio tipico è quello in cui si conservano dati relativi a "soggetti", quali il fatto di essere o no sposati. In un record "variante" per prima viene definita la parte fissa, poi la parte "variante", adoperando un apposito selettore (di qualsiasi tipo semplice) e usando le parole riservate CASE e OR. Per esempio:

```
TYPE
   genere = (maschio, femmina);
   variant = RECORD
        {tutti i campi comuni}
        CASE sposato : boolean OF
           false : ( );
           true  : (DataMatr :string;
                CASE sesso : genere OF
                   maschio : ( );
                   femmina : (NomeDaRagazza :string))
        END; {variant}
```

Anche per le liste relative a campi vuoti si devono usare le parentesi. L'occupazione nel file è fissa (in base alla variante più lunga), ma si può risparmiare memoria ricorrendo a puntatori di varianti, quali new(p, true, maschio).
Le poche debolezze del PASCAL sono state largamente eliminate dagli esperti di questo linguaggio (e dallo stesso Wirth nel suo MODULA-2). Si tratta quindi di un linguaggio estremamente potente, pur rimanendo di dimensioni contenute, efficiente, adatto ad applicazioni generiche e di facile apprendimento. Ovviamente, se si desidera aderire allo standard ISO, può darsi che alcune operazioni a livello di sistema non siano possibili, ma la redazione di speciali routine in codice macchina o in BCPL, in seguito accodandole con l'apposito linker ai programmi in PASCAL è solo una delle tante soluzioni. Tuttavia, uno dei pregi maggiori del PASCAL è proprio quello di poter redigere programmi trasportabili da un apparecchio all'altro.

Il Tandy 1000

Anatomia di un microcomputer a diffusione europea: tecnologie a confronto

Benché la Tandy sia stata una tra le prime ditte ad entrare nel mercato dei microcomputer (con il TRS-80), non è riuscita ad ottenere l'enorme successo delle sue concorrenti iniziali: la Apple e la Commodore. Negli ultimi tempi, però, la Tandy ha dimostrato una ferma intenzione di rilanciare la propria immagine anche nel settore degli apparecchi ad uso commerciale. Da una parte, assicuratasi la collaborazione della ACT (produttrice dell'Apricot), la Tandy ha rimodernato i modelli già in commercio sul mercato europeo. Dall'altra ha introdotto notevoli innovazioni anche nel proprio settore produttivo.

Si è sempre affermato che la ditta in grado di produrre un apparecchio effettivamente compatibile con il PC della IBM, ma ad un prezzo "stracciato", avrebbe dominato la scena sia del mercato commerciale, sia di quello degli home computer. Ciò vale soprattutto negli USA, dove i potenziali consumatori tendono ad acquistare per uso domestico apparecchi il cui prezzo è considerato alto per l'equivalente settore europeo.

La Tandy afferma che il Modello 1000 è perfettamente compatibile con l'IBM, ma viene venduto ad un prezzo incredibilmente inferiore.

La versione base possiede 128 Kbyte di memoria e una singola unità a dischetti, ma il modello 1000 qui esaminato è quello con 256 Kbyte e due unità a dischetti.

A prima vista, il sistema assomiglia non poco al PC della IBM: l'unità centrale è racchiusa in un robusto e compatto contenitore, che incorpora anche le interfacce e le unità a dischetti; il monitor viene appoggiato su detto contenitore e la tastiera, indipendente, viene collocata nella posizione più comoda per l'uso.

Anche la tastiera assomiglia a quella del PC IBM, con una leggera incurvatura dei tasti per facilitare la scrittura anche ad alta velocità. Inoltre, per meglio posizionare la tastiera, esistono due piedini retrattili che servono a dare un'inclinazione di circa 15°.

La disposizione dei tasti è diversa da quella della tastiera IBM, sebbene siano presenti tutti quelli che ne garantiscono la compatibilità. Questa scelta ha i suoi pro e i suoi contro. Indubbiamente la tastiera IBM è stata criticata da molti per la scomoda posizione di alcuni importanti tasti, quali l'Alternate e lo Shift, e le discutibili dimensioni dei tasti Control e Return.

Di ciò, evidentemente, deve aver tenuto conto la Tandy nel progettare la propria tastiera. Il tasto '/', il cui uso è ostacolato usando lo Shift di sinistra, è stato rimosso del tutto e la sua funzione si ottiene premendo Shift e i tasti 4 e 7 del tastierino numerico laterale. I tasti Alternate



CHRIS STEVENS

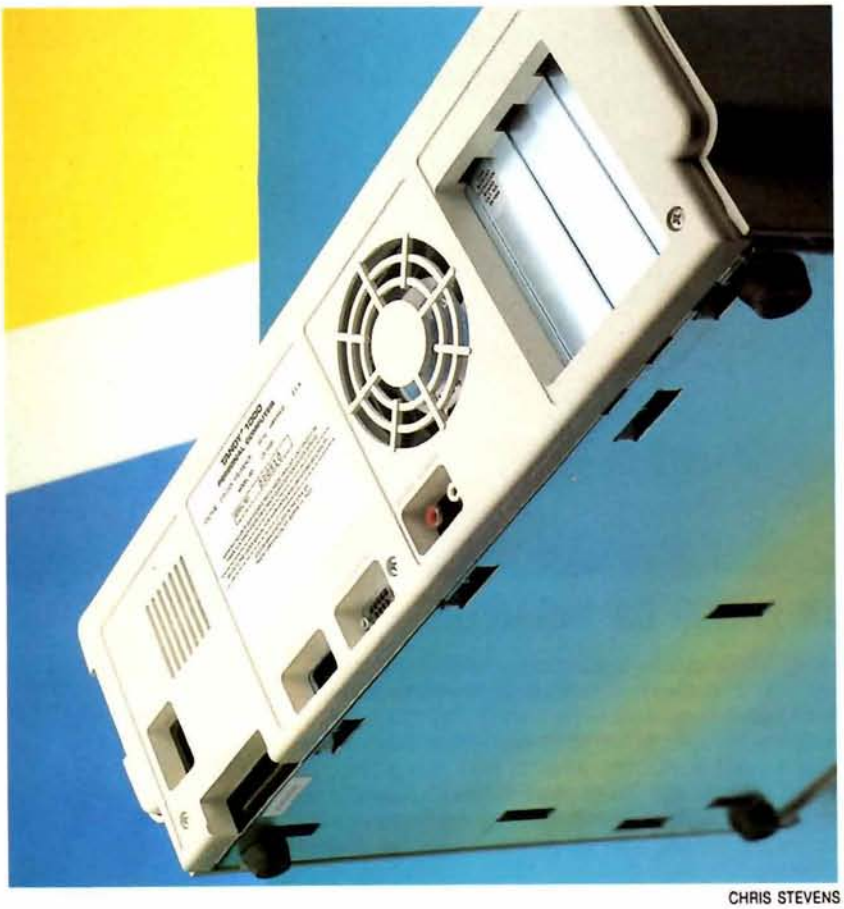
UN PASSO DA GIGANTE

Il Tandy 1000, un computer compatibile con il PC IBM, è il tentativo della Tandy di riguadagnare il proprio spazio nel mercato dei micro. Per ottenere la compatibilità è stata usata la CPU 8088 e unità a dischetti da 5 1/4 pollici. La foto mostra il modello con doppia unità a dischetti, dotato di un'espansione da 128 Kbyte. Sullo schermo si possono vedere le immagini del 'Flight Simulator' prodotto dalla Microsoft Flight per il Personal IBM. La possibilità di far girare questo programma è una delle prove più ardue da superare per un IBM-compatibile

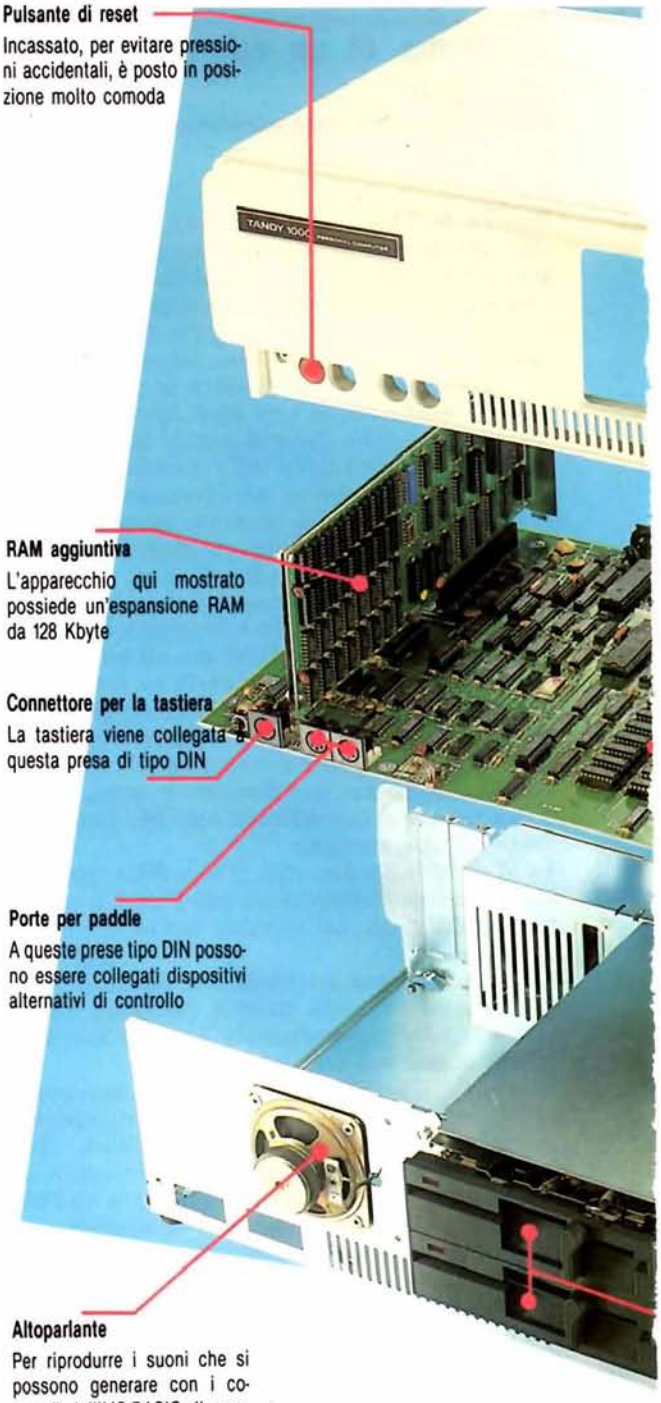
e Caps Lock sono stati collocati, rispettivamente, all'estrema sinistra e all'estrema destra della tastiera. Queste modifiche rendono certamente più agevole la digitazione che non sulla tastiera IBM. I tasti hanno una corsa comoda e sono di ottima qualità.
L'unico inconveniente, peraltro abbastanza raro, può presentarsi per chi si sia abituato alla tastiera IBM e passi poi al Tandy.
Sono state apportate anche altre modifiche alla tastiera: i tasti Insert e Delete sono stati abbinati alle funzioni + e – e si trovano sulla parte superiore della tastiera, anziché su quella inferiore. Tasti addizionali includono quelli riservati al controllo del cursore, a Hold, a Print e a Break, tutti situati tra la tastiera principale e quella numerica. Questa disposizione ha obbligato a cambiar posto ai tasti di funzione, che nella tastiera Tandy si trovano sul bordo superiore. Questa posizione non è forse tra le migliori ma, in compenso, ha permesso di aggiungere due tasti di funzione in più.
Passando alle caratteristiche dell'unità centrale, si nota che il contenitore ha grosso modo le stesse dimensioni di quello del PC IBM. Sul lato frontale, a sinistra, ci sono le due unità a dischetti da 5 1/4 pollici, benché la versione base ne incorpori una sola. Apparentemente meno rumorose che sull'IBM, le unità a dischetti dimostrano tempi di accesso leggermente diversi. Un fatto spiacevole è che l'estrazione dei dischetti non è a molla: occorre tirar fuori il dischetto a mano.
Sotto la bombatura del pannello frontale, vicino ad una fessura per l'aereazione, si trova una coppia di connettori DIN a sei poli, da usare per joystick o altri dispositivi di controllo esterno. A sinistra di questi c'è un grosso pulsante arancione, da usare per il 'reset' del sistema. Nonostante le dimensioni è difficile azionarlo inavvertitamente, poiché

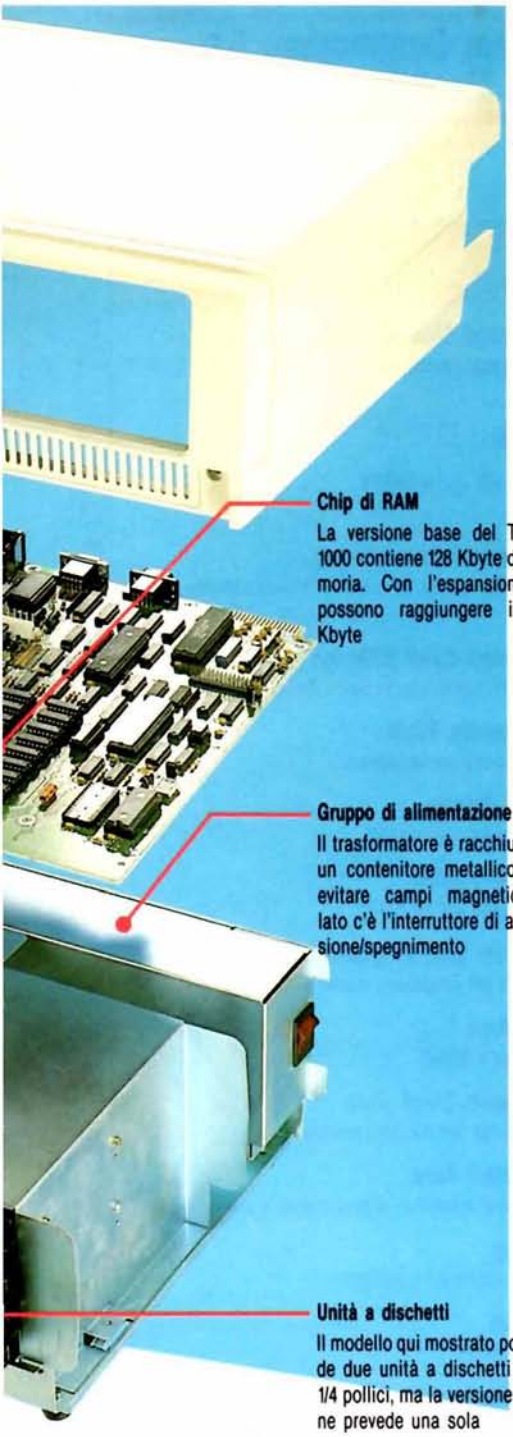
I connettori del Tandy

Oltre ad avere un'interfaccia parallela Centronics, il Tandy possiede anche due prese per monitor con uscite sia RGB sia a video composito. Esiste anche una presa audio, grazie alla quale si possono riprodurre i suoni generati dal computer su di un impianto hi-fi



CHRIS STEVENS





Chip di RAM
La versione base del Tandy 1000 contiene 128 Kbyte di memoria. Con l'espansione si possono raggiungere i 640 Kbyte

Gruppo di alimentazione
Il trasformatore è racchiuso in un contenitore metallico, per evitare campi magnetici. A lato c'è l'interruttore di accensione/spegnimento

Unità a dischetti
Il modello qui mostrato possiede due unità a dischetti da 5 1/4 pollici, ma la versione base ne prevede una sola

è incassato nell'involucro. In effetti, una simile collocazione del tasto di reset è particolarmente felice, se si considera che da molti programmi applicativi occorre uscire proprio agendo su tale tasto.
Sul lato posteriore sono disposte le interfacce per periferiche. Alla sinistra dell'apparecchio c'è l'ingresso per l'alimentazione, che si trova subito sopra il connettore "a pettine" per la porta parallela tipo Centronics. A destra c'è un connettore D a sette poli per penna ottica e, più oltre, una porta a sette poli per il collegamento di un monitor RGB. Ancora più a destra, sotto la ventola di raffreddamento, c'è una coppia di prese "microjack": una fornisce l'uscita video per monitor monocromi o a colori, l'altra emette i segnali audio prodotti dal computer, che possono essere riprodotti da un impianto ad alta fedeltà. All'estrema destra si trovano tre connettori per espansioni, grazie ai quali può essere aggiunta memoria o possono venir collegate ulteriori periferiche.

Meriti a confronto

La Tandy ha puntato molto sul fatto che il proprio apparecchio offre la possibilità di un maggior numero di interfacce rispetto al PC della IBM. Sebbene ciò sia vero, nella versione base il numero di interfacce è il minimo indispensabile per un apparecchio commerciale. La memoria standard è, ad esempio, di 128 Kbyte, insufficiente per molti dei più recenti package di software integrato. Sebbene il Tandy 1000 sia un apparecchio da postazione singola, l'interfaccia seriale RS232 (opzionale) è ormai divenuta essenziale per qualsiasi comunicazione con altri computer.
D'altro canto, il prezzo per aggiungere queste opzioni è decisamente inferiore a quello necessario per espandere un IBM.
Rimane da considerare la compatibilità del software con il PC della IBM. Il problema, qui, non è fornire l'apparecchio di una CPU 8088 o di un sistema operativo MS-DOS (per questo basta accordarsi con i relativi distributori), bensì di riprodurre fedelmente il comportamento delle routine del BIOS (Basic Input/Output System) dell'IBM, che è coperto da copyright. Infatti molti programmi applicativi progettati per l'IBM adoperano chiamate dirette al BIOS e, se le routine non si trovano al medesimo indirizzo, il loro funzionamento può essere parzialmente compromesso.
A questo proposito, il BIOS del Tandy 1000 (scritto dalla Phoenix Compatibility Corporation) è a dir poco eccezionale. Non solo funziona egregiamente il *Lotus 1-2-3* (anche se con 256 Kbyte si trova un po' stretto), ma anche il *Wordstar 2000* e il *dBase II*.
C'è da notare, però, che il Tandy 1000 è relativamente più lento del PC IBM per quanto concerne i calcoli aritmetici in BASIC. Un conto da 0 a 1000, usato come banale prova, richiede circa sei secondi: un tempo superiore perfino a certi computer a 8 bit!
Tutto sommato, il Tandy 1000 è da considerarsi una valida ed economica alternativa per l'utente commerciale che desideri avvalersi dell'enorme quantità di software disponibile per il PC della IBM.

TANDY 1000

DIMENSIONI

mm. 420 x 335 x 150

CPU

8088, funzionante a 4,77 MHz

MEMORIA

128 Kbyte di RAM, espandibile a 640 Kbyte

VIDEO

80 x 25 caratteri nel modo testuale.
640 x 200 pixel nel modo grafico ad alta risoluzione. 8 colori visualizzabili simultaneamente, da una gamma di 16

INTERFACCE

Uscite per monitor a video composito e RGB; porta per stampante; uscita audio e tre connettori per interfaccia

LINGUAGGI FORNITI

BASIC, oltre a tutti i linguaggi sotto MS-DOS (COBOL, FORTRAN, ecc.)

TASTIERA

Tipo QWERTY con 90 tasti, inclusi 12 di funzione e tastierina numerica indipendente

DOCUMENTAZIONE

I manuali spiegano l'uso del sistema operativo MS-DOS, del MS-BASIC e del package Deskmate, fornito assieme all'apparecchio

PRO

Totalmente compatibile con il PC IBM, con in più la possibilità di aggiung ere interfacce non disponibili sulla versione base dell'altro apparecchio a un prezzo altamente competitivo

CONTRO

Tecnicamente, il progetto del Tandy non porta innovazioni rispetto al PC della IBM (se si escludono le modifiche alla tastiera). L'impiego della CPU 8088, anziché della più recente 8086 comporta tempi operativi più lunghi

DATABASE

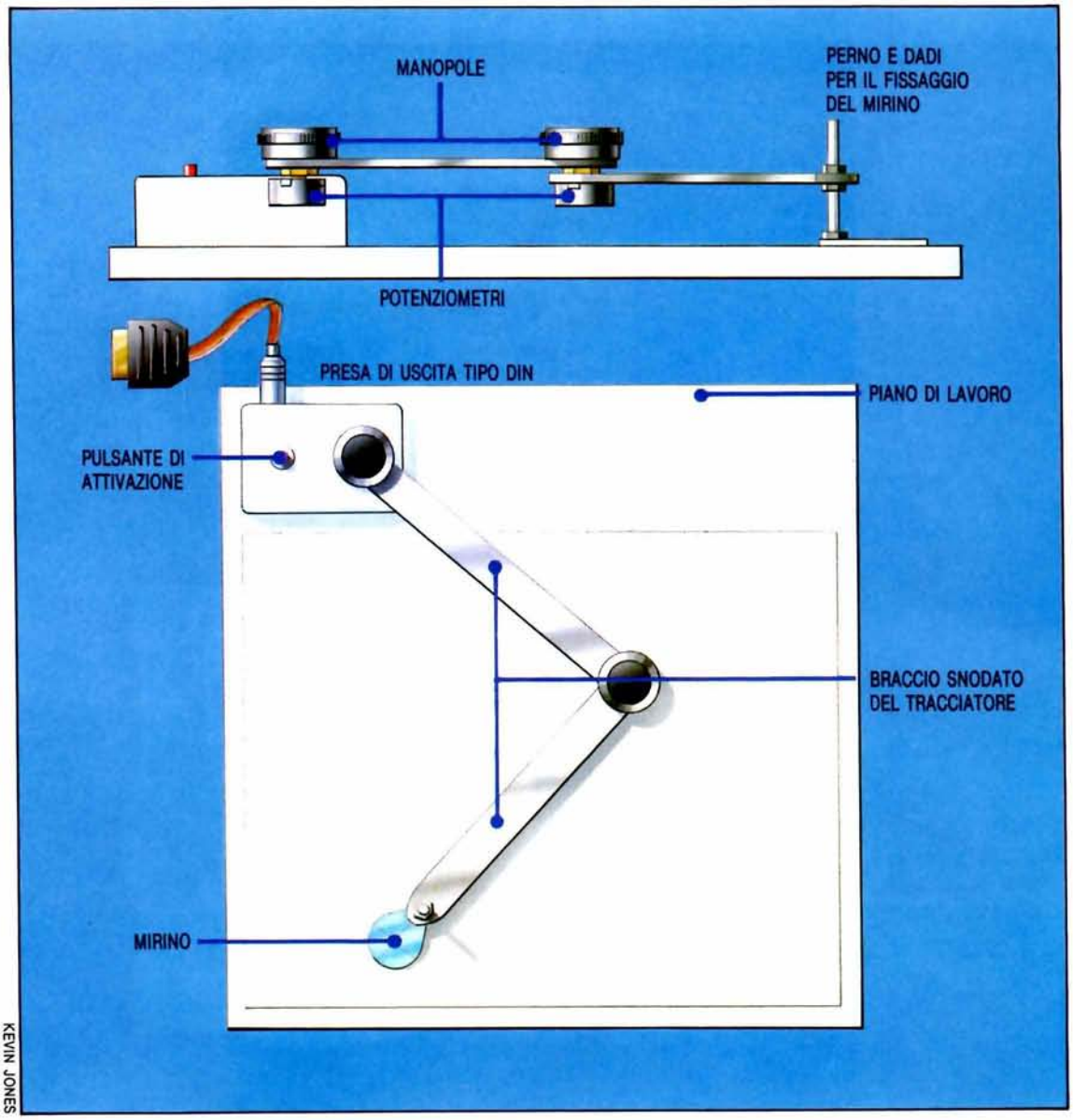
La settima tabella della mappa di memoria del Commodore 64. Per concessione della Commodore Business Machines

INDIRIZZO ESADECIMALE	ETICHETTA	LOCAZIONE DECIMALE	DESCRIZIONE
00D6	TBLX	214	**Current Cursor Physical Line Number** Numero di linea fisica occupata dal cursore
00D7		215	**Temp Data Area** Area temporanea per dati
00D8	INSRT	216	**Flag: Insert Mode, >0 = #INSTs** Flag: Modo Insert attivo
00D9-00F2	LDTB1	217-242	**Screen Line Link Table / Editor Temps** Tabella di collegamento per le linee dello schermo / locazioni temporanee per l'Editor
00F3-00F4	USER	243-244	**Pointer: Current Screen Color RAM loc.** Puntatore: locazione RAM dell'attuale schermo dei colori
00F5-00F6	KEYTAB	245-246	**Vector: Keyboard Decode Table** Vettore: Tabella di decodifica per la tastiera
00F7-00F8	RIBUF	247-248	**RS-232 Input Buffer Pointer** Puntatore al buffer d'ingresso per RS-232
00F9-00FA	ROBUF	249-250	**RS-232 Output Buffer Pointer** Puntatore al buffer di uscita per RS-232
00FB-00FE	FREKZP	251-254	**Free 0-Page Space for User Programs** Spazio libero in pagina 0 per programmi utente
00FF	BASZPT	255	**BASIC Temp Data Area** Area Dati temporanea per il BASIC
0100-01FF		256-511	**Micro-Processor System Stack Area** Area per lo stack di sistema del microprocessore
0100-010A		256-266	**Floating to String Work Area** Area di lavoro utilizzata per numeri in virgola mobile e stringhe
0100-013E	BAD	256-318	**Tape Input Error Log** Segnalatore di errore di lettura su registratore
0200-0258	BUF	512-600	**System INPUT Buffer** Buffer di sistema per la INPUT
0259-0262	LAT	601-610	**KERNAL Table: Active Logical File No's** Tabella KERNAL: Numeri dei file logici attivi
0263-026C	FAT	611-620	**KERNAL Table: Device No. for Each File** Tabella KERNAL: Numero del dispositivo abbinato a ciascun file
026D-0276	SAT	621-630	**KERNAL Table: Second Address Each File** Tabella KERNAL: Indirizzo secondario di ciascun file
0277-0280	KEYD	631-640	**Keyboard Buffer Queue (FIFO)** Coda per il buffer di tastiera (FIFO)
0281-0282	MEMSTR	641-642	**Pointer: Bottom of Memory for O.S.** Puntatore: Indirizzo più basso di memoria per il sistema operativo

Un tracciatore digitale

La prima parte di un nuovo progetto hardware da usare con il Micro BBC

Un tracciatore digitale è un dispositivo che consente, tramite l'impiego di una tavoletta e di un braccio snodato, di riprodurre disegni sullo schermo del computer. L'immagine, una volta convertita in segnali digitali, può essere memorizzata su nastro o dischetto ed essere riutilizzata in seguito. Il componente principale del dispositivo è un braccio metallico, formato da due sezioni incernierate tra loro e con il piano di lavoro per formare un "gomito" e una "spalla". Su ciascuno dei due snodi è fissato un potenziometro che, variando l'ampiezza dei segnali elettrici a seconda della posizione assunta, fornisce i dati da inviare al computer. Chi possiede un Micro BBC è privilegiato, in questo progetto, poiché questo computer ha un convertitore analogico/digitale incorporato. Ciò significa che il segnale in uscita da ciascun potenziometro può essere applicato direttamente a questo convertitore, che provvede a produrre gli impulsi digitali necessari al computer per determinare l'esatta posizione della "penna" (o "mirino") del tracciatore.
Le illustrazioni mostrano come sono disposti e fissati i principali elementi del tracciatore. L'unità è fissata su di una tavoletta di legno quadrata, che misura 46 cm. per lato. Su questo piano di lavoro viene appoggiato il disegno da riprodurre. Il braccio snodabile è fissato a una scatoletta di plastica, a sua volta fissata al pia-



KEVIN JONES

Elementi traccianti

Il digital tracer qui descritto impiega due potenziometri per fornire le informazioni necessarie a individuare la posizione del "mirino". Tali informazioni vengono poi trasformate, passando attraverso il convertitore analogico/digitale, ed elaborate dal software che provvede a riprodurre i disegni sullo schermo o a memorizzarli su dischetto

no di lavoro. Sulla scatoletta è montata una presa tipo DIN a 5 poli, sulla quale viene innestato il cavetto di collegamento con la porta analogica del computer, e un pulsante usato come semplice comando per attivare o disattivare il tracciamento. Sull'estremità libera del braccio è fissato un dischetto di plastica trasparente, sul quale viene incisa una piccola croce che rappresenta il "mirino" per seguire i contorni dei disegni. Il sistema di montaggio consente anche di lavorare su materiali di vario spessore e non solo su sottili fogli di carta.

ELENCO COMPONENTI

Quantità	Descrizione
2	potenziometri lineari 100 K Ohm
2	manopole per potenziometri 40 mm.
1	pulsante normalmente aperto
1	scatola in plastica mm. 114x76x38
1	spinotto DIN a 5 poli
1	presa DIN a 5 poli
1 m.	cavo piatto a 10 poli
1	confezione di viti M5
1	confezione di viti M3
1	confezione di dadi M3
1	confezione di viti autofilettanti
1	connettore D a 15 poli maschio
1	connettore D a 15 poli femmina

Varie

1	striscia di alluminio mm. 600x25x2,5 (*)
1	tavoletta rivestita in formica o melamina cm. 46 x46
1	pezzetto di plastica trasparente (**)

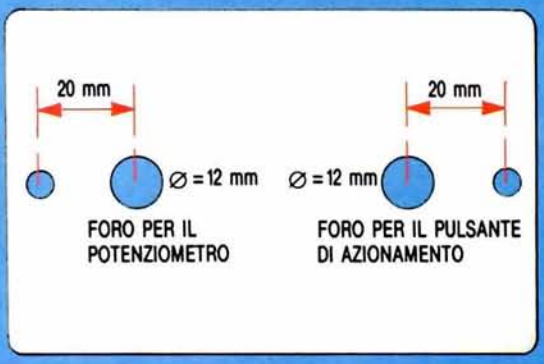
Note

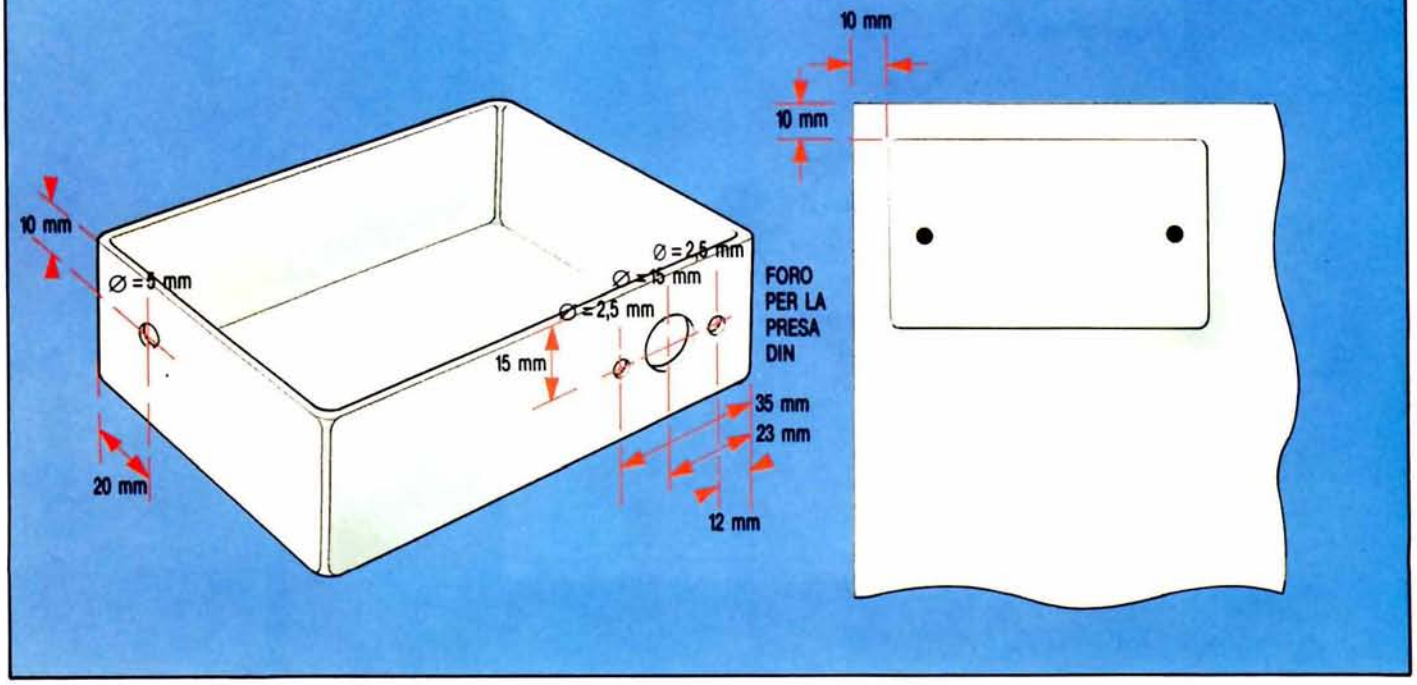
(*) In alternativa pu essere usata una striscia di legno

(**) Può essere usata la parte trasparente di una vecchia custodia per musicassette

FASE 1: FORATURA DELLA SCATOLETTA

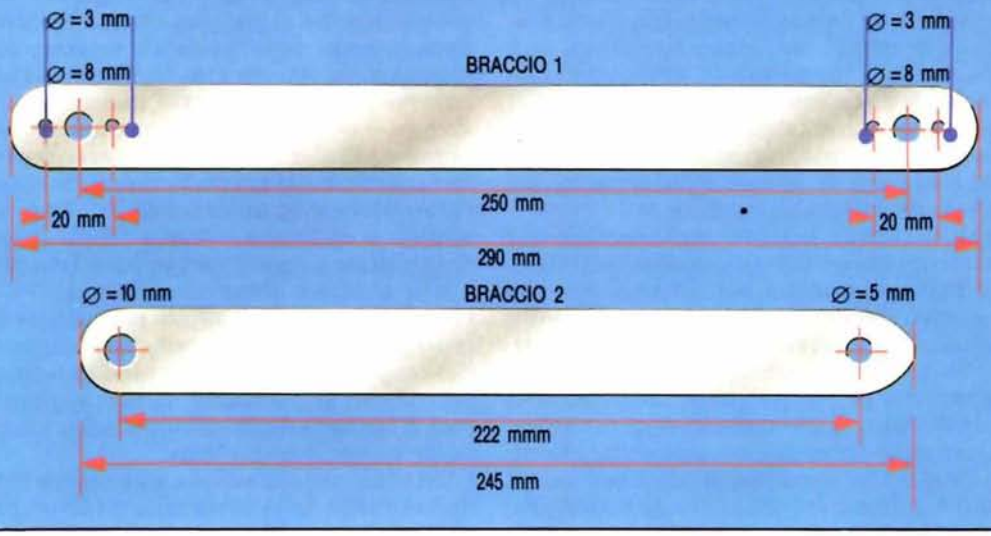
Il primo passo, nella costruzione del tracciatore digitale, se necessario, è quello di tagliare il piano di lavoro delle dimensioni volute. Poi occorre forare la piccola scatoletta di supporto del braccio, come indicato in figura. Infine, prima di fissare definitivamente la scatoletta sul piano di lavoro (lasciando un margine di circa 10 mm. tutto attorno), è necessario montare il braccio, la presa DIN e il pulsante





FASE 2: LE SEZIONI DEL BRACCIO

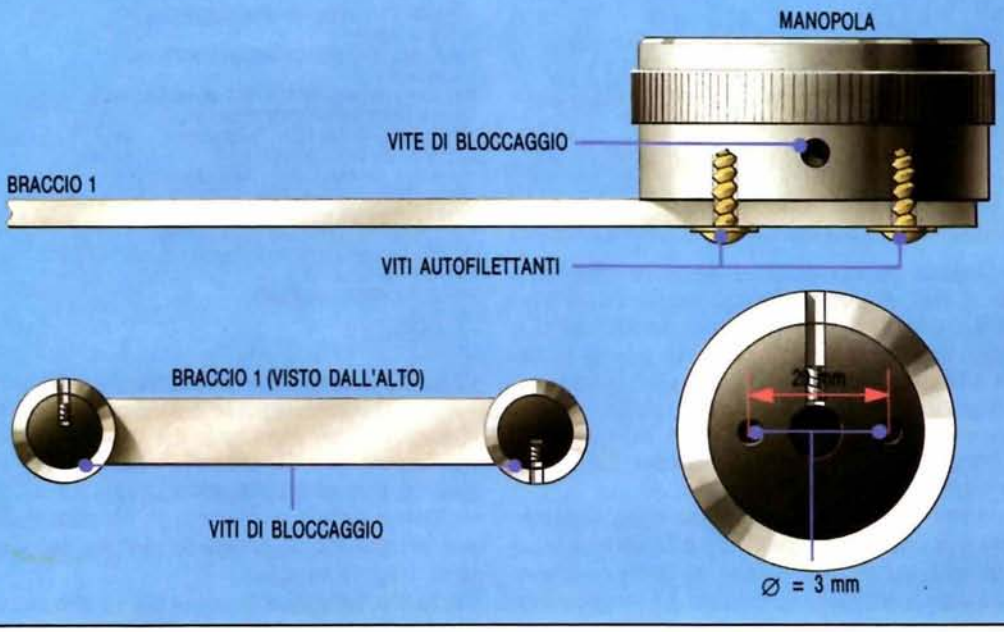
La striscia di alluminio va tagliata in due sezioni, che vanno forate come illustrato in figura. Si noti che, nonostante le misure non siano estremamente critiche, le distanze tra i fori praticati su ciascuna estremità devono essere precise. Queste distanze, infatti, vengono adoperate dal software per eseguire i calcoli necessari per individuare la posizione del mirino. Prima di forare, quindi, si controlli l'esattezza di queste misure



FASE 3: FISSAGGIO DELLE MANOPOLE

L'ultima operazione, per il momento, è quella di montare le manopole (simili a quelle per il controllo del volume nelle radio) a ciascuna estremità del Braccio 1. Queste manopole servono per assicurare il fissaggio dei perni mobili dei potenziometri, tramite un'apposita piccola vite. Si noti che la posizione delle vitine di bloccaggio dei perni deve essere contrapposta, come indicato nella figura in basso a sinistra.

Le manopole vengono fissate all'asta del braccio praticando due piccoli fori (diametro mm. 3) nella base delle manopole e facendoci passare due viti autofilettanti. Queste vanno avvitate con attenzione: abbastanza perché l'insieme sia solido, ma non tanto da rovinare l'impanatura delle viti



KEVIN JONES

Disegni reticolari

Esploriamo le routine in virgola mobile del BASIC per il Commodore 64

Sfortunatamente, le routine in virgola mobile dell'interprete BASIC per il Commodore 64 non sono affatto ben documentate nei vari manuali. Ad esempio, non viene riprodotta una tabella dei salti (come quella del kernel) che permetta di accedere facilmente ad esse. Il progetto presentato in questa prima parte dedicata all'argomento, riguarda il tracciamento e la rotazione di disegni sullo schermo ad alta risoluzione del Commodore 64.
Gli accorgimenti e le teorie matematiche qui esposte, comunque, possono essere fruttuosamente impiegate anche per ottenere routine aritmetiche più veloci (ad esempio per la moltiplicazione di matrici numeriche). Si rammenti, però, che non sempre conviene eseguire calcoli matematici in tempo reale, specialmente quando si tratta di tracciare disegni in rapida sequenza sullo schermo: spesso conviene eseguire i calcoli in precedenza, memorizzare i risultati e, infine, passare al disegno animato. Ai nostri fini, tuttavia, il calcolo in tempo reale è il più adatto.
Le variabili del BASIC sono conservate, in memoria, sopra il programma BASIC. L'indirizzo iniziale della tabella delle variabili è contenuto in un puntatore che si trova nelle locazioni 45 e 46 (decimale). Come tutti i puntatori nel Commodore 64, anche questo è conservato nel formato basso/alto. Pertanto la formula per ricavare l'indirizzo della tabella è:

PEEK(45) + 256*PEEK(46)

I puntatori associati alle variabili, assieme al loro normale contenuto, sono qui riassunti:

Puntatore	Funzione	Normale contenuto
43/44	Inizio del BASIC	2049
45/46	Inizio variabili	Dipende dalla lunghezza del programma
47/48	Inizio delle matrici	Dipende dal numero di variabili
49/50	Fine matrici + 1	Dipende dal numero e dalle dimensioni delle matrici
51/52	Fondo area stringhe	Dipende dal numero e dalla lunghezza delle stringhe
55/56	Limite sup. memoria	40960

Le stringhe, gestite dinamicamente, crescono verso il basso, partendo dal limite superiore della memoria via via che vengono definite. Le matrici, invece, sono conservate sopra la tabella delle variabili così quando nel programma viene definita una nuova variabile, il sistema operativo sposta l'intera area delle matrici di tanti byte quanti ne occorrono per far entrare la nuova variabile.
La conservazione delle stringhe è necessariamente più complessa rispetto alle altre variabili. Infatti, le variabili intere (se non appartengono a una matrice) e le variabili a virgola mobile occupano sette byte, ma una stringa può occuparne fino a 255. Per superare questa complicazione, il sistema operativo conserva nella tabella delle variabili soltanto la lunghezza della stringa e un puntatore all'indirizzo dove essa è conservata in memoria. Se la stringa è definita nel programma BASIC (ad esempio: A$ = "ABCD") allora il puntatore contiene l'indirizzo del primo byte della stringa nell'area del programma BASIC. Una simile stringa è chiamata "statica". Se e quando il programma altera il contenuto di una stringa, allora questa è chiamata "dinamica" e viene conservata nell'area subito sottostante il limite superiore della memoria. Ovviamente, l'indirizzo del puntatore nella tabella delle variabili riflette la posizione di tale stringa. Solo così il formato degli elementi nella tabella rimane costante (sette byte).
L'indirizzo di una stringa può essere estratto dalla tabella delle variabili usando un piccolo trucco e poche istruzioni BASIC. Infatti, subito dopo l'uso di una variabile BASIC, le locazioni in pagina zero 71 e 72 (decimali) contengono l'indirizzo di tale variabile. Tuttavia il recupero di questo indirizzo deve essere immediato, poiché il sistema operativo adopera la locazione 71 per certe operazioni aritmetiche.
Il seguente programma mostra il formato usato per conservare in memoria le variabili ordinarie. La prima routine estrae una stringa dalla memoria e conserva immediatamente il valore del puntatore in 71 e 72 depositandolo in due locazioni del buffer per il registratore. In seguito l'indirizzo salvato viene usato per calcolare quello della stringa.

```
1000 REM** TROVA UNA STRINGA IN MEMORIA**
1010 X$ = "ABCDEF"
1020 REM** X$** DIVENTA VARIABILE CORRENTE
1030 X$ = X$ + ""
1040 REM** SALVA IL PUNTATORE ALLA TABELLA **
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM** INDIRIZZO NELLA TABELLA **
1070 INDR = PEEK(828) + 256*PEEK(829)
1080 REM** ESAMINA ELEM. IN TABELLA **
1090 LS = PEEK(INDR):REM LUNGH. STRINGA
1100 SA = PEEK(INDR + 1) + 256*PEEK(INDR + 2)
1110 REM SA = INDIRIZZO INIZ. DELLA STRINGA
1120 REM ** ADESSO LEGGE LA STRINGA **
1130 FOR I = SA TO SA + LS
1140 VAR$ = VAR$ + CHR$(PEEK(I))
1150 NEXT
1160 PRINT VAR$
```

Qualcuno potrebbe pensare che adoperando variabili intere (quelle con il simbolo %) si risparmierebbe memoria e i calcoli sarebbero più rapidi. Sul Commodore 64, ciò non è vero poiché, per eseguire calcoli sugli interi, questi vengono prima convertiti in virgola mobile e poi vengono richiamate le routine per il calcolo in virgola mobile!
Pertanto, sebbene le variabili intere occupino

soltanto due byte (salvo facciano parte di una matrice), ad esse vengono assegnati comunque sette byte di memoria e quelli in eccesso vengono semplicemente ignorati nel corso del calcolo. La seguente routine consente di individuare in memoria una variabile intera:

```
1000 REM•• TROVA UN INTERO IN MEMORIA••
1010 X%=3456
1020 REM•• RENDE X% VARIABILE CORRENTE ••
1030 X%=X%
1040 REM•• SALVA IL PUNTATORE ••
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM•• INDIRIZZO IN TABELLA VARIAB.••
1070 INDR=PEEK(828)+256*PEEK(829)
1080 REM•• ESAMINA ELEM. IN TABELLA ••
1090 LO=PEEK(INDR+1):HI=PEEK(INDR)
1100 REM•• CALCOLA RISULTATO ••
1110 BITSEGN=(HI AND 128)/128
1120 VAR=LO+256*(HI AND 127)—32768*BITSEGN
1130 PRINT VAR
```

La stessa tecnica di individuazione può essere impiegata per localizzare variabili a virgola mobile. Tuttavia, esiste un metodo molto più economico per far ciò, basato sul fatto che, usando DEF FN per definire una funzione della variabile X, tale variabile viene usata (pur rimanendo inalterata) ad ogni chiamata della funzione.
Questo programma adopera DEF FN per calcolare l'indirizzo della corrente variabile BASIC, sempre ricorrendo alle locazioni 71 e 72. Poiché X è usata come variabile per la funzione, si può esser certi che l'indirizzo risultante è quello del primo byte nella tabella delle variabili. Viene quindi chiamata la funzione FN, per assegnare l'indirizzo a INDR. Si noti che passando uno zero (o un altro valore) attraverso il comando FN, nella tabella non cambia né il valore di X né il suo indirizzo.

```
1000 REM•• TROVA VARIABILE A V.M. ••
1010 DEF FNINDR(X)=PEEK(71)+256*PEEK(72)
1020 INDR=FNINDR(0):REM FORNISCE SEMPRE L'INDIRIZZO DI X
1030 X=—3.14159
1040 REM•• CONVERTE DA BASE 2 A DECIMALE••
1050 POTENZA=2PEEK(INDR)—129)
1060 SEGNO=(—1)PEEK(INDR+1)AND128)/128)
1070 REM•• PARTE FRAZIONARIA LUNGA 31 BIT••
1080 D1=PEEK(INDR+1)AND127:REM 7 BIT
1090 D2=PEEK(INDR+2):REM 8 BIT
1100 D3=PEEK(INDR+3):D4=PEEK(INDR+4)
1110 REM •• GULP! ••
1120 FRAZ=2↑(—7)*D1+2↑(—15)*D2+2↑(—23)*D3+2↑(—31)*D4
1130 MANT=1+FRAZ
1140 VAR=SEGNO*POTENZA*MANT
1150 PRINT VAR
```

Matrici in memoria

Quando il BASIC esegue un'istruzione DIM, viene riservata memoria per la matrice specificata. Il blocco di memoria consiste in una "testata" più il numero di byte necessari per la conservazione degli elementi. Il formato degli elementi varia a seconda dei tipi di variabile esaminati fin qui (vedere diagramma). Comprendere quali siano le differenze è molto importante se si intende accedere agli elementi della matrice usando il codice macchina.
Inoltre occorre esaminare anche il meccanismo del calcolo in virgola mobile. Quando l'interprete BASIC esegue calcoli in virgola mobile, eventuali risultati intermedi vengono depositati in due "accumulatori a virgola mobile", generalmente chiamati FAC e ARG. Il formato è identico ai numeri in virgola mobile conservati in memoria. FAC si trova agli indirizzi da $61 a $65 (da 97 a 101 decimale) e ARG da $69 a $6D (da 105 a 109 decimale). Per semplicità, in seguito vengono adoperate soltanto le routine dell'interprete che scambiano numeri tra FAC e la memoria. Le routine dell'interprete BASIC con le quali familiarizzarsi, in questa prima parte, sono:

- MOVFM (indirizzo $BBA2)

Questa routine copia in FAC un numero a virgola mobile che si trova in memoria. Simbolicamente, ciò è rappresentato con F←M. Per chiamare la routine, occorre caricare nell'accumulatore il byte basso dell'indirizzo per il numero in memoria e nel registro Y il byte alto.

- MOVMF (indirizzo $BBD4)

Questa routine deposita il contenuto di FAC in sette byte consecutivi di memoria. Simbolicamente: M←F. Per la chiamata occorre caricare nei registri X e Y, rispettivamente, il byte basso e quello alto dell'indirizzo di memoria ove depositare il numero.

- FMULT (indirizzo $BA28)

Questa è la routine di moltiplicazione: essa moltiplica il contenuto di FAC per un altro numero che si trova in memoria, depositando il risultato in FAC. Il primo numero viene caricato in FAC usando MOVFM, mentre il byte basso e quello alto vanno posti, rispettivamente, nei registri X e Y prima di chiamare questa routine. Il risultato viene depositato in FAC, ma può essere trasferito in memoria usando MOVMF.

- FADD (indirizzo $B867)

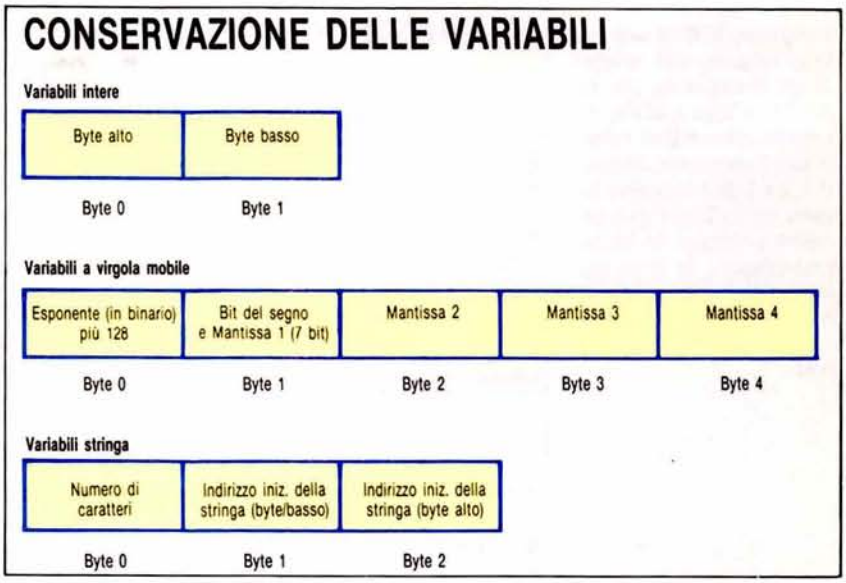
Questa routine esegue la somma FAC=MEM+FAC. Per chiamarla, presupponendo che in FAC sia già presente un valore, occorre depositare nell'accumulatore e nel registro Y(U), rispettivamente, il byte basso e quello alto di MEM.

- FSUB (indirizzo $B850)

Questa routine esegue la differenza FAC=MEM—FAC. La chiamata si esegue come descritto per FADD.

Queste sono le routine dell'interprete adope-

CONSERVAZIONE DELLE VARIABILI

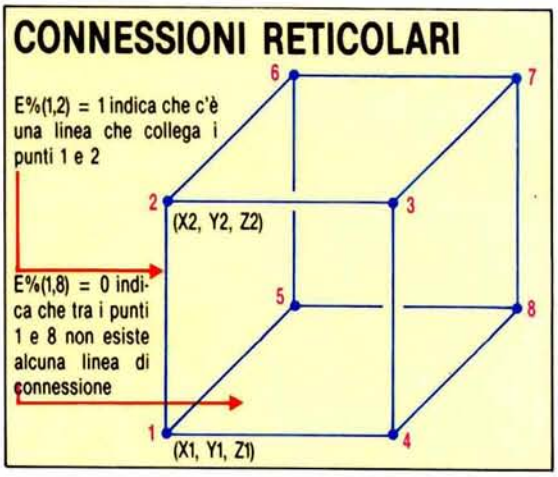


Tipi di variabili

Sul Commodore 64 esistono tre tipi di variabili, ognuno dei quali ha un diverso formato quando conservato in memoria. Le variabili intere sono conservate in due byte, sotto forma di numeri in complemento a due. I numeri a virgola mobile (o fluttuante) richiedono sette byte: cinque per la mantissa e due per il segno e per l'esponente. Le variabili stringa sono conservate in una diversa zona di memoria, usando un byte per ogni carattere della stringa. Nella tabella delle matrici, però, non vengono depositati questi dati, bensì tre byte: uno per specificare la lunghezza della stringa e due che forniscono l'indirizzo a 16 bit al quale è memorizzata la stringa

Incroci d'effetto

Il programma ROTSUB ruota una figura reticolare sullo schermo ad alta risoluzione del Commodore 64. La figura è definita impiegando quattro matrici, tre delle quali contengono le coordinate X, Y e Z di ciascun nodo. La quarta matrice E%(,) è usata per indicare se due punti del reticolo sono collegati o no da un segmento. Nell'esempio qui a fianco le quattro matrici sono inizializzate per produrre la forma di un cubo



rate nella formulazione della prima parte dei programmi di grafica reticolare.

Il concetto di base del nostro progetto grafico è che una figura reticolare può essere definita per mezzo di una serie di numeri (o nodi) e una matrice di connessioni. I nodi hanno coordinate X(I),Y(I) e Z(I), dove I varia da 1 a NP (il numero di punti). La matrice delle connessioni è E%(I,J), dove I e J variano entrambi da 1 a NP. L'elemento E%(I,J) vale 1 se il punto I è unito da un tratto al punto J, altrimenti vale zero. Questo metodo non è tra i più economici in fatto di memoria, poiché si usano due byte quando in realtà ne basterebbe uno solo, tuttavia facilita la definizione dei punti collegati tra loro. Inoltre, nella pratica, NP non sarà mai molto grande.

Per far comunicare il BASIC con la routine in codice macchina che esegue la rotazione, si usano delle PEEK con le quali depositare gli indirizzi delle matrici, iniziando dall'elemento 1. Pertanto, prima di tutto occorre trovare gli indirizzi di X(1), Y(1), Z(1), e E%(1,1). Al codice macchina, inoltre, viene passato il numero di punti, NP, e il valore di COS e SIN dell'angolo di rotazione attorno all'asse Z desiderato.

È bene che qualsiasi progetto in codice macchina proceda per piccoli passi, altrimenti il compito di individuare eventuali errori, in seguito, diventa impossibile. Il progetto è stato così suddiviso in tre parti. Nella prima vengono definiti gli algoritmi e redatta una versione di prova in BASIC, che ruota un cubo tridimensionale attorno all'asse Z, proiettando i risultati sul piano X e Y.

L'obiettivo, poi, sarà quello di convertire in codice macchina il programma BASIC. Anche questo avviene gradualmente: per prima viene convertita la subroutine che inizia alla linea 1800. Il risultato è il listato I-ROTSUB, per il quale è fornito anche il relativo programma di prova.

Nella seconda e ultima parte dedicata a questo argomento, viene completato il programma in codice macchina per la rotazione di immagini reticolari e fornito il relativo programma caricatore in BASIC.

PROGRAMMA I-ROTSUB 64

Il seguente listato Assembly va assemblato e convertito in codice con l'apposito programma di load. Si assegni al file risultante il nome "I-ROT.HEX"

```
;++++++++++++++++++++++++++++++
;++       I-ROTSUB 64        ++
;++                          ++
;++++++++++++++++++++++++++++++
;++  USA MATRICI DEFINITE    ++
;++  DAL BASIC. ESEGUIRE     ++
;++  POKE DEGLI INDIRIZZI    ++
;++++++++++++++++++++++++++++++
;
;PLTSUB = $C183
;XLO    = $C103
;XHI    = $C104
;YLO    = $C105
;
*       = $C544
;
;+++ VARIABILI ROTSUB +++
;
; VARIABILI CHIAMATE DAL BASIC
XBASLO  *=*+1  ;  POKE50500,X(0)LO
XBASHI  *=*+1  ;  POKE50501,X(0)HI
YBASLO  *=*+1  ;  POKE50502,Y(0)LO
YBASHI  *=*+1  ;  POKE50503,Y(0)HI
NP      *=*+1  ;  POKE50504,NP
; VARIABILI USATE DAL C/M
XILO    *=*+1
XIHI    *=*+1
YILO    *=*+1
YIHI    *=*+1
CSLO    *=*+1
CSHI    *=*+1
SNLO    *=*+1
SNHI    *=*+1
MEM1    *=*+5  ;VAR. VIRGOLA MOBILE
MEM2    *=*+5  ;VAR. VIRGOLA MOBILE
FAC     =$0061
ARG     =$0069
;
;ROUTINE ARITMETICHE DELL'INTERPRETE
;
FMULT   =$BA28  ; FAC=FAC*ARG
FADDT   =$B86A  ; FAC=FAC+ARG
FSUB    =$B850  ; FAC=MEM-FAC
FADD    =$B867  ; FAC=FAC+MEM
MOVFM   =$BBA2  ; FAC=MEMORIA
MOVMF   =$BBD4  ; MEMORIA=FAC
;
;+++ SALVA I  REGISTRI +++
;
        PHA
        TXA
        PHA
        TYA
        PHA
;
;+++ INIZIALIZZA VARIABILI +++
;
        LDA XBASLO
        STA XILO
        LDA XBASHI
        STA XIHI
        LDA YBASLO
        STA YILO
        LDA YBASHI
        STA YIHI
;
;+++ ESEGUE MEM1=X(I)*CS-Y(I)*SN +++
;
INIZIO
        LDA XILO
        LDY XIHI
        JSR MOVFM  ; FAC = X(I)
        LDA CSLO
        LDY CSHI
        JSR FMULT  ; FAC = X(I)*CS
        LDX #<MEM1
        LDY #>MEM1
        JSR MOVMF  ; MEM1=X(I)*CS
        LDA YILO
        LDY YIHI
        JSR MOVFM  ; FAC = Y(I)
        LDA SNLO
        LDY SNHI
        JSR FMULT  ; FAC = Y(I)*SN
        LDA #<MEM1
        LDY #>MEM1
        JSR FSUB   ; FAC = MEM1-FAC
        LDX #<MEM1
        LDY #>MEM1
        JSR MOVMF  ; MEM1 = FAC
;
;+++ ESEGUE MEM2=Y(I)*CN+X(I)*SN +++
;
        LDA YILO
        LDY YIHI
        JSR MOVFM  ; FAC = Y(I)
        LDA CSLO
        LDY CSHI
        JSR FMULT  ; FAC = Y(I)*CS
        LDX #<MEM2
        LDY #>MEM2
        JSR MOVMF  ; MEM2 = Y(I)*CS
        LDA XILO
        LDY XIHI
        JSR MOVFM  ; FAC = X(I)
        LDA SNLO
        LDY SNHI
        JSR FMULT  ; FAC = X(I)*SN
        LDA #<MEM2
        LDY #>MEM2
        JSR FADD   ; FAC = MEM2+FAC
        LDX #<MEM2
        LDY #>MEM2
        JSR MOVMF  ; MEM2= FAC
;
;+++ ESEGUE X(I) =MEM1:Y(I)=MEM2 +++
;
        LDA #<MEM1
        LDY #>MEM1
        JSR MOVFM  ; FAC =MEM1
        LDX XILO
        LDY XIHI
        JSR MOVMF  ; X(I)=FAC
        LDA #<MEM2
        LDY #>MEM2
        JSR MOVFM  ; FAC =MEM2
        LDX YILO
        LDY YIHI
        JSR MOVMF  ; Y(I)=FAC
;
;+++ VERIFICA DI FINE CICLO +++
;
        DEC NP
        BEQ EXIT
;
;+++ INCREMENTA PUNTATORI ALLE MATRICI +++
;
        LDA #$05
        CLC
        ADC XILO
        STA XILO
        BCC XNOHI
        INC XIHI
XNOHI
        LDA #$05
        CLC
        ADC YILO
        STA YILO
        BCC YNOHI
        INC YIHI
YNOHI
        JMP INIZIO
;
;+++ RECUPERA REGISTRI DALLO STACK +++
;
EXIT    PLA
        TAY
        PLA
        TAX
        PLA
;
        RTS
END.
```

PROGRAMMA DI ROTAZIONE DEL CUBO IN BASIC

```
1000 REM** CUBO RUOTANTE IN BASIC **
1010 IF A=0 THEN A=1: LOAD"PLOTSUB.HEX",8,1
1020 IF A=1 THEN A=2: LOAD"LINESUB.HEX",8,1
1030 REM** DIMENSIONA MATRICI **
1040 NP=8: REM NUMERO DI PUNTI
1050 DIM X(NP), Y(NP), Z(NP)
1060 DIM ED(NP,NP): REM CONNESSIONI DEI CONTORNI
1070 REM**INIZIALIZZA LE MATRICI **
1080 REM-- DATI SULLE COORDINATE DEL CUBO
1090 DATA  75, 75, 75: REM----------/1
1100 DATA -75, 75, 75: REM 4 PUNTI  /2
1110 DATA -75,-75, 75: REM SUPERIORI/3
1120 DATA  75,-75, 75: REM----------/4
1130 DATA  75, 75,-75: REM----------/5
1140 DATA -75, 75,-75: REM 4 PUNTI  /6
1150 DATA -75,-75,-75: REM INFERIORI/7
1160 DATA  75,-75,-75: REM----------/8
1170 REM** RUOTA IL CUBO SU ASSE-X DI  π/4
1180 FOR I=1 TO NP
1190 READ X(I), Y(I), Z(I)
1200 Y(I)=Y(I)*COS( π/4)-Z(I)*SIN( π/4)
1210 Z(I)=Z(I)*COS( π/4)+Y(I)*SIN( π/4)
1220 NEXT
1230 REM** RUOTA IL CUBO SU ASSE Z DI   π/4
1240 FOR I=1 TO NP
1250 X(I)=X(I)*COS( π/4)-Y(I)*SIN( π/4)
1260 Y(I)=Y(I)*COS( π/4)+X(I)*SIN( π/4)
1270 NEXT
1280 REM-- DATI SU CONNESSIONI TRA BORDI --
1290 E(1,2)=1: REM 1 CONNESSO A 2
1300 E(2,3)=1: E(3,4)=1: E(4,1)=1
1310 E(5,6)=1: REM QUADRATO INFERIORE
1320 E(6,7)=1: E(7,8)=1: E(8,5)=5
1330 E(5,1)=1: REM BORDI VERTICALI
1340 E(6,2)=1: E(7,3)=1: E(8,4)=1
1350 REM** RENDE E(I,J) SIMMETRICA **
1360 FOR I=1 TO NP: FORJ=1 TO NP
1370 IF E(I,J)=1 THEN E(J,I)=1
1380 NEXT:NEXT
1390 REM****************************
1400 REM** TRACCIA CUBO RUOTANTE **
1410 SA=2* π/45
1420 FOR A= π/4 TO  π/4 + 2* π  STEP SA
1430 GOSUB 1800: REM RUOTA CON PASSO SA
1440 GOSUB 1590: REM INIZ/PULISCE SCHERMO
1450 REM-- TRACCIA IL CUBO --
1460 FOR I=1 TO NP
1470 FOR J=1 TO I
1480 IF E(I,J)=0 THEN 1510: REM NON CONNESSI
1490 GOSUB 1630: REM CALCOLA PROIEZIONE
1500 GOSUB 1670: REM CONGIUNGE PUNTI
1510 NEXT:NEXT
1520 REM---------------------------
1530 NEXT A: REM ANGOLO SUCCESSIVO
1540 REM****************************
1550 REM** ATTENDE **
1560 GET A$: IF A$="" THEN 1560
1570 GOSUB 1760: REM RESET DELLO SCHERMO
1580 END
1590 REM ** PREDISPONE HI-RES **
1600 POKE 49408,1: POKE 49409,1
1610 POKE 49410,1: SYS 49422
1620 RETURN
1630 REM** CALCOLA PROIEZIONE IN HI-RES **
1640 X1%=X(I)+159: Y1%=199-(Z(I)+100)
1650 X2%=X(J)+159: Y2%=199-(Z(J)+100)
1660 RETURN
1670 REM** LINESUB **
1680 IF (X1%=X2%) AND (Y1%=Y2%) THEN RETURN
1690 MHI=INT(X1%/256): MLO=X1%-256*MHI
1700 NHI=INT(X2%/256): NLO=X2%-256*NHI
1710 POKE 49920,MLO: POKE 49921,MHI
1720 POKE 49922,NLO: POKE 49923,NHI
1730 POKE 49924,Y1%: POKE 49925,Y2%
1740 SYS 49934: REM LINESUB
1750 RETURN
1760 REM** RESET DELLO SCHERMO **
1770 POKE 49408,0: SYS 49422
1780 PRINT CHR$(147)
1790 RETURN
1800 REM** RUOTA IL CUBO SU ASSE Z DI SA **
1810 FOR I=1 TO NP
1820 X(I)=X(I)*COS(SA)-Y(I)*SIN(SA)
1830 Y(I)=Y(I)*COS(SA)+X(I)*SIN(SA)
1840 NEXT
1850 RETURN
```

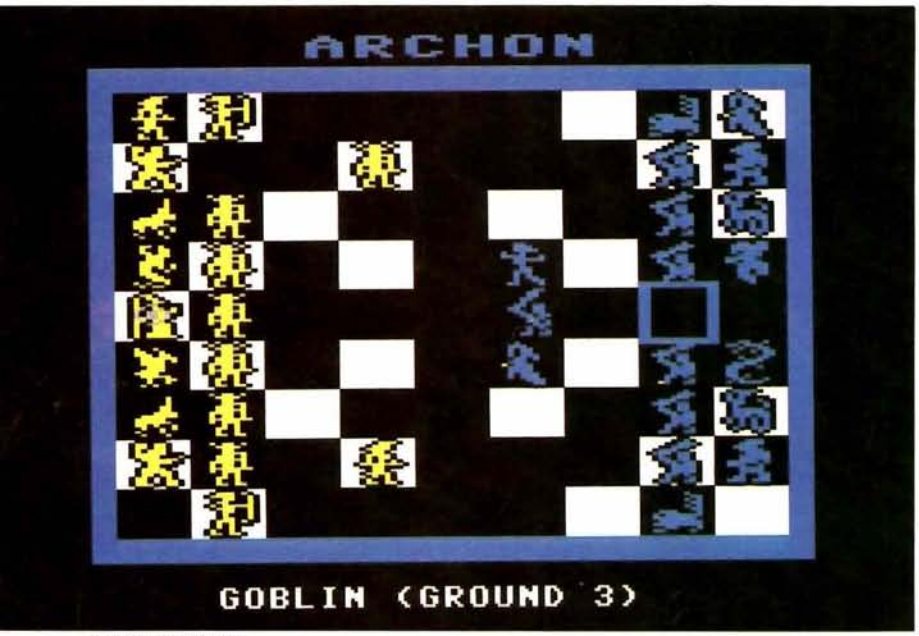
PROGRAMMA DI VERIFICA PER I-ROTSUB

Il seguente programma in BASIC va trascritto e memorizzato sotto il nome "TEST I-ROT". Si noti che nessuna variabile va definita nella sezione che va dalla linea 1880 alla 2010, dove è situata la chiamata SYS50523, altrimenti gli indirizzi di base delle matrici vengono alterati e il programma produce risultati imprevedibili

```
1000 REM** TEST DI I-ROTSUB **
1010 IF A=0 THEN A=1: LOAD"PLOTSUB.HEX",8,1
1020 IF A=1 THEN A=2: LOAD"LINESUB.HEX",8,1
1030 IF A=2 THEN A=3: LOAD"I-ROT.HEX",8,1
1040 REM** DIMENSIONA MATRICI **
1050 NP=8: REM NUMERO DI PUNTI
1060 DIM X(NP), Y(NP), Z(NP)
1070 DIM ED(NP,NP): REM CONNESSIONI DEI CONTORNI
1080 REM ** INIZIALIZZA LE MATRICI **
1090 REM--DATI SULLE COORDINATE DEL CUBO
1100 DATA  75, 75, 75: REM----------/1
1110 DATA -75, 75, 75: REM 4 PUNTI  /2
1120 DATA -75,-75, 75: REM SUPERIORI/3
1130 DATA  75,-75, 75: REM----------/4
1140 DATA  75, 75,-75: REM----------/5
1150 DATA -75, 75,-75: REM 4 PUNTI  /6
1160 DATA -75,-75,-75: REM INFERIORI/7
1170 DATA  75,-75,-75: REM----------/8
1180 REM** ROTAZIONE ATTORNO ASSE X DI  π/4
1190 FOR I=1 TO NP
1200 READ X(I), Y(I), Z(I)
1210 Y(I)=Y(I)*COS( π/4)-Z(I)*SIN( π/4)
1220 Z(I)=Z(I)*COS( π/4)+Y(I)*SIN( π/4)
1230 NEXT
1240 REM** ROTAZIONE ATTORNO ASSE Z DI  π/4
1250 FOR I=1 TO NP
1260 X(I)=X(I)*COS( π/4)-Y(I)*SIN( π/4)
1270 Y(I)=Y(I)*COS( π/4)+X(I)*SIN( π/4)
1280 NEXT
1290 REM--DATI SU CONNESSIONI TRA BORDI--
1300 E(1,2)=1: REM 1 CONNESSO A 2
1310 E(2,3)=1: E(3,4)=1: E(4,1)=1
1320 E(5,6)=1: REM QUADRATO INFERIORE
1330 E(6,7)=1: E(7,8)=1: E(8,5)=5
1340 E(5,1)=1: REM BORDI VERTICALI
1350 E(6,2)=1: E(7,3)=1: E(8,4)=1
1360 REM** RENDE E(I,J) SIMMETRICA **
1370 FOR I=1 TO NP: FORJ=1 TO NP
1380 IF E(I,J)=1 THEN E(J,I)=1
1390 NEXT:NEXT
1400 REM****************************
1410 REM** TRACCIA CUBO RUOTANTE **
1420 SA=2*  π/45: CS=COS(SA): SN=SIN(SA)
1430 FOR A=0 TO 2* π  STEP SA
1440 GOSUB 1870: REM RUOTA CON PASSO SA
1450 GOSUB 1600: REM INIZ/PULISCE SCHERMO
1460 REM-- TRACCIA IL CUBO --
1470 FOR I=1 TO NP
1480 FOR J=1 TO I
1490 IF E(I,J)=0 THEN 1520: REM NON CONNESSI
1500 GOSUB 1640: REM CALCOLA PROIEZIONE
1510 GOSUB 1680: REM CONGIUNGE PUNTI
1520 NEXT:NEXT
1530 REM-------------------
1540 NEXT A: REM ANGOLO SUCCESSIVO
1550 REM**************************
1560 REM** ATTENDE **
1570 GET A$: IF A$="" THEN 1570
1580 GOSUB 1770: REM RESET DELLO SCHERMO
1590 END
1600 REM ** PREDISPONE HI-RES **
1610 POKE 49408,1: POKE 49409,1
1620 POKE 49410,1: SYS 49422
1630 RETURN
1640 REM** CALCOLA PROIEZIONE IN HI-RES **
1650 X1%=X(I)+159: Y1%=199-(Z(I)+100)
1660 X2%=X(J)+159: Y2%=199-(Z(J)+100)
1670 RETURN
1680 REM** LINESUB **
1690 IF (X1%=X2%) AND (Y1%=Y2%) THEN RETURN
1700 MHI=INT(X1%/256): MLO=X1%-256*MHI
1710 NHI=INT(X2%/256): NLO=X2%-256*NHI
1720 POKE 49920,MLO: POKE 49921,MHI
1730 POKE 49922,NLO: POKE 49923,NHI
1740 POKE 49924,Y1%: POKE 49925,Y2%
1750 SYS 49934: REM LINESUB
1760 RETURN
1770 REM** RESET DELLO SCHERMO **
1780 POKE 49408,0: SYS 49422
1790 PRINT CHR$(147)
1800 RETURN
1810 REM** RUOTA CUBO SU ASSE Z DI SA
1820 FOR I=1 TO NP
1830 X(I)=X(I)*CS-Y(I)*SN
1840 Y(I)=Y(I)*CS+X(I)*SN
1850 NEXT
1860 RETURN
1870 REM** RUOTA ATTORNO ASSE Z DI SA
1880 X(1)=X(1): REM X(1) VAR CORRENTE
1890 POKE 50500,PEEK(71): REM X(1)LO
1900 POKE 50501,PEEK(72): REM X(1)HI
1910 Y(1)=Y(1): REM Y(1) VAR CORRENTE
1920 POKE 50502,PEEK(71): REM Y(1)LO
1930 POKE 50503,PEEK(72): REM Y(1)HI
1940 POKE 50504,NP: REM NUMERO DI PUNTI
1950 CS=CS: REM CS VAR CORRENTE
1960 POKE 50509,PEEK(71)
1970 POKE 50510,PEEK(72)
1980 SN=SN: REM SN VAR CORRENTE
1990 POKE 50511,PEEK(71)
2000 POKE 50512,PEEK(72)
2010 SYS 50523
2020 RETURN
```

"Archon"

Nato da una combinazione tra gli scacchi e la grafica da arcade, ecco un originale gioco



DIMENSION GRAPHICS

Di casella in casella

"Archon" può anche essere descritto come un gioco di scacchi animato. Quando si ordina a un pezzo di muoversi, lo spostamento avviene con modalità diverse, a seconda del tipo di pezzo. Una volta giunto nella casella desiderata, se questa è occupata da un avversario, inizia un combattimento in puro stile arcade. Il risultato del duello dipende sia dall'abilità dei giocatori sia dalla forza di ciascun pezzo coinvolto

Archon, prodotto dalla Ariolasoft, è un gioco che può affascinare sia l'appassionato di scacchi sia il più accanito frequentatore di sale giochi.

Alla base del gioco c'è la lotta tra le forze della Luce e del Buio. Il gioco inizia mostrando una scacchiera nella quale gli avversari sono individuati secondo righe e colonne, in modo molto simile al gioco degli scacchi. I pezzi che appartengono alla Luce hanno nomi quali "phoenix" e "knight", mentre quelli del Buio sono chiamati "goblins", "dragons" e via dicendo.

Ogni icona, corrispondente ai pezzi, ha una propria forza e capacità di movimento. Alcuni pezzi, ad esempio, possono "volare" (ossia saltare i pezzi che si parano di fronte), altri possono muoversi soltanto sul terreno.

La scacchiera, che assomiglia a quella per gli scacchi, è suddivisa in una matrice di nove per nove caselle, tuttavia il colore delle varie caselle può variare dal bianco al nero o viceversa durante il gioco. Ciò dipende dal fatto che le forze della Luce sono più potenti sulle caselle bianche e le forze del Buio sulle caselle nere.

L'obiettivo, in *Archon*, è di occupare le cinque caselle "del potere" avversarie: quattro sono disposte a croce sui bordi della scacchiera. All'apertura del gioco, la strategia consigliata è quella di portare i propri pezzi su caselle del colore appropriato, in modo da conferire loro maggior forza e renderle meno vulnerabili. Così facendo, inoltre, si aprono le file per consentire il movimento ai pezzi retrostanti, che non possono volare, e per costruire così la propria barriera difensiva.

Fin qui il gioco ha fortissime somiglianze con gli scacchi, ma le differenze si fanno evidenti non appena si tenta di occupare una casella in cui si trova un pezzo avversario: anziché "mangiarlo" direttamente, viene offerta una visione ravvicinata della casella; questa diventa un campo di battaglia tra i due avversari, che si battono in puro stile arcade. La forza di ciascun pezzo è raffigurata da una barretta che, colpita dall'avversario, decresce progressivamente.

Se la barretta scompare, l'icona abbinata è sconfitta e il possesso della casella passa al vincitore.

Poiché le modalità del combattimento variano a seconda delle icone coinvolte, il duello può essere talvolta anche impari, data la superiorità di uno dei due pezzi sull'altro. Per esempio, un drago può lanciare fiamme da una parte all'altra della casella, mentre un cavaliere deve avvicinarsi molto all'avversario per usare la propria spada. Il combattimento è reso più impegnativo dalla presenza di barriere che mutano di posizione durante lo scontro.

Ciascuno dei due giocatori dispone di un pezzo "magico": un mago per la Luce e una strega per il Buio. Le malie lanciate da questi pezzi speciali, che hanno uguale forza ma possono essere usate una sola volta, possiedono qualità staordinarie: possono far resuscitare un particolare pezzo in una casella a scelta o invocare creature chiamate "elementali", utili quando si tratta di affrontare un nemico molto più forte. La creatura elementale, però, scompare al termine del duello e non occupa caselle.

Il gioco può esser giocato tra due persone oppure da un solo giocatore contro il computer. Benché il computer, come avversario, sia uno stratega mediocre, la sua abilità nel condurre il combattimento tipo arcade è notevole e vincere non è affatto facile.

Archon: Per Commodore 64
Editori: Ariolasoft
Autori: Anne Westfall, Jon Freeman, Paul Reiche III
Joystick: Necessari
Formato: Cassetta

Stampanti per Spectrum



CHRIS STEVENS

Una scelta ragionata

Negli ultimi anni sono apparse sul mercato numerose stampanti termiche. Alcune, come la Floyd 40 e l'Alphacom 32, sono specificamente destinate allo Spectrum Sinclair mentre altre, come l'Epson P40 e la Brother HR-5, possono essere collegate a numerosi microcomputer

Quattro diverse alternative alla stampante ZX

In genere la prima periferica abbinata a un computer è la stampante, che permette di produrre su carta listati dei propri programmi. Per chi possiede uno Spectrum Sinclair, però, tale scelta pone numerosi problemi: da una parte lo Spectrum non possiede nessuna delle interfacce convenzionali, abitualmente usate per le stampanti, dall'altra la ZX della Sinclair Research, oggi fuori commercio, produce stampati di bassa qualità, che tendono a sbiadirsi col tempo. Poiché la Sinclair non ha intenzione di migliorare la sua stampante, altri fabbricanti hanno prodotto stampanti proprie a prezzi concorrenziali.

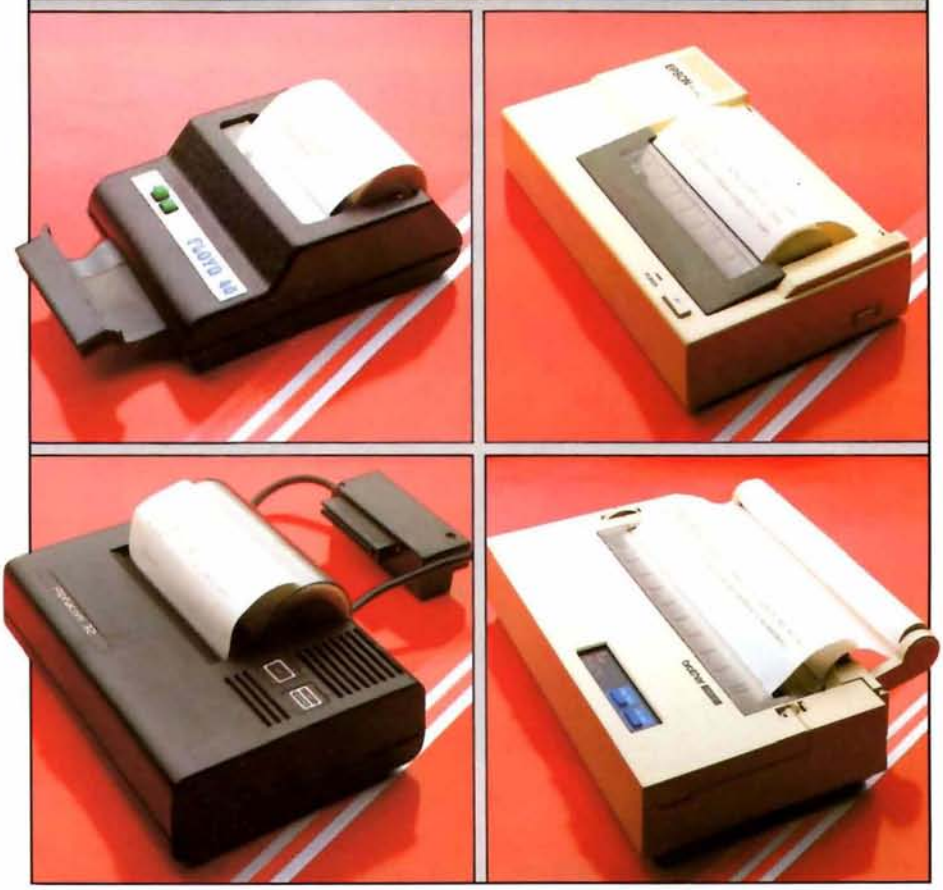
Vengono qui esaminate alcune di queste alternative, ma l'acquisto di una stampante richiede considerazioni preliminari. Innanzi tutto bisogna valutare il costo complessivo: possono esserci spese extra non comprese nel prezzo di listino dell'apparecchio, come quella di un'interfaccia RS232C da inserire tra lo Spectrum e la stampante, ad esempio; in tal caso, per usarla, va acquistata l'Interface 1, che fornisce allo Spectrum una porta RS232C.

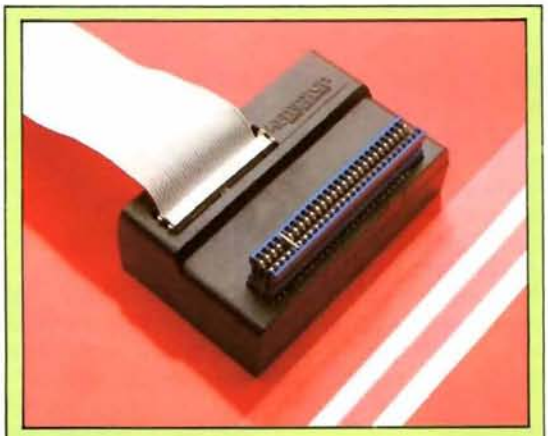
Spese extra

Molte stampanti funzionano solo con particolari tipi di carta dal costo piuttosto elevato. Per le stampanti termiche quali la ZX Printer, ad esempio, occorre una carta speciale, sensibile al calore. Quindi, acquistando un apparecchio di questo tipo, non ci si può difendere contro eventuali aumenti del prezzo della carta. Inoltre conviene assicurarsi che il rivenditore dell'apparecchio garantisca una regolare fornitura di carta e di altri accessori. Poiché molte parti meccaniche della stampante sono mobili (ad esempio, la testina di scrittura), esse sono maggiormente soggette

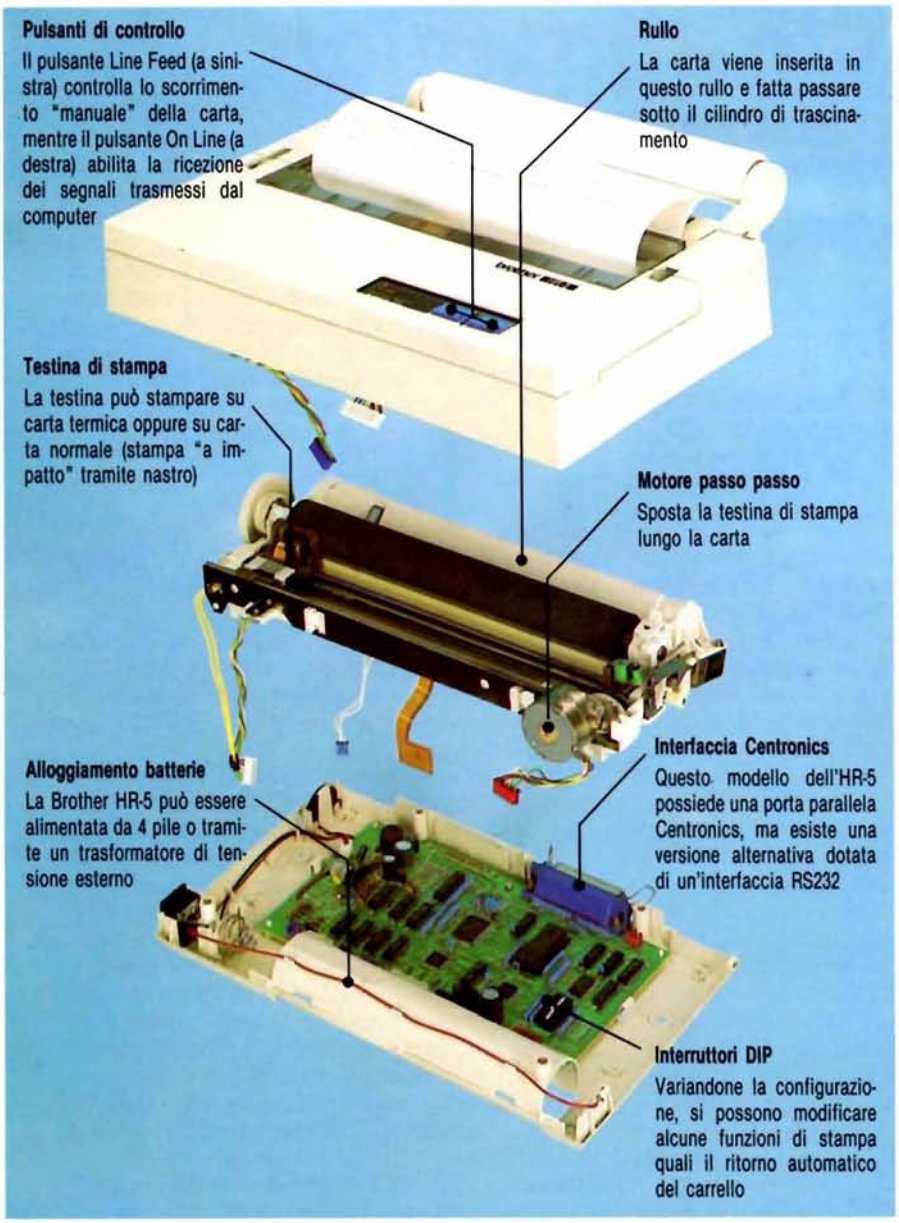
ad usura rispetto alle componenti elettroniche: conviene acquistare l'apparecchio da rivenditori che garantiscano un adeguato servizio di assistenza.
Infine, va tenuto presente il problema della compatibilità software. Tutte le stampanti sono programmate per rispondere a codici (di solito i caratteri ASCII) che ne controllano le funzioni di base come il ritorno-carrello, il numero di caratteri per linea e il formato della stampa. Benché molte di queste funzioni siano identiche a quelle usate dallo Spectrum, alcune stampanti possono risultare incompatibili.
Come altre case costruttrici di microcomputer, la Sinclair ha infatti adattato alcuni dei codici ASCII alle proprie esigenze: in alcuni casi, addirittura, il manuale indica che un codice ha un particolare significato mentre, in realtà, lo Spectrum lo interpreta diversamente. Quindi, prima dell'acquisto, è consigliabile eseguire alcune prove di stampa per verificare la compatibilità tra la stampante in questione e lo Spectrum.
Le stesse precauzioni valgono per il software che si intende usare. Può infatti accadere che alcuni codici di comando, che funzionano alla perfezione con programmi BASIC, non possano essere usati con un particolare pacchetto di word processing. Ci si può quindi trovare nella poco invidiabile situazione di dover salvare un testo su disco o nastro, dover spegnere il sistema e, infine, rileggere il file in modo sequenziale per poterlo stampare.
Basandosi sui criteri di scelta appena esposti, esaminiamo alcune delle stampanti termiche offerte dal mercato. Tra queste, la Alphacom 32 è la più economica e quella che più assomiglia alla ZX Printer: possiede infatti gli stessi caratteri tipografici e l'identico formato massimo della stampa a 32 colonne.
I progettisti hanno dotato l'Alphacom 32 di un sistema d'alimentazione proprio, che però risulta superfluo in quanto il bus d'espansione dello Spectrum fornisce una tensione di 9 volt, sufficiente per alimentare una stampante termica.
La struttura della Floyd 40, fabbricata dalla Shiva Marketing, appare piuttosto delicata. La custodia in materiale plastico è fragile e il perno che sostiene il rotolo di carta è un piccolo cilindretto di legno. Nonostante ciò, l'apparecchio viene considerato migliore dell'Alphacom. La Floyd 40 viene alimentata direttamente dal bus d'espansione dello Spectrum e usa una carta bianca che rende più leggibili i caratteri stampati in nero. Comunque, ciò che la pone al di sopra delle precedenti stampanti è la possibilità di accettare una serie di caratteri di controllo che modificano il formato di stampa.
I caratteri di controllo vengono inviati alla stampante tramite normali comandi LPRINT, seguiti da virgolette. Inoltre, ogni carattere speciale va racchiuso tra due punti esclamativi, per comunicare alla stampante che si tratta di un comando e non di un carattere da stampare. L'istruzione LPRINT "!H!", ad esem-

MODELLO	Alphacom 32	Floyd 40	Epson P40	Brother HR-5
ALIMENTAZIONE	A batteria	Esterna	Esterna	Esterna/a batteria
STAMPA	Termica	Termica	Termica	Termica/su nastro
INTERFACCIA USATA	Bus d'espansione	Bus d'espansione	RS232 o parallela Centronics	RS232 o parallela Centronics





Interfaccia Kempston
Per poter collegare lo Spectrum ad alcune delle stampanti più sofisticate (dotate di interfaccia RS232 o Centronics), occorre dotare il sistema di un'apposita interfaccia. L'interfaccia Kempston, ad esempio, consente di abbinare lo Spectrum a varie stampanti con compatibilità Centronics



CHRIS STEVENS

pio, ordina alla Floyd 40 di stampare caratteri a doppia altezza. La ripetizione del comando annulla tale formattazione.
Altre opzioni disponibili riguardano la grafica, il "wordwrap", la stampa a caratteri inversi e a doppia larghezza (espansi). La testina termica, a matrice di 5x7 punti, produce discrete immagini nel modo grafico e la qualità di stampa è paragonabile a quella di apparecchi molto più costosi. La Floyd 40 è piuttosto lenta, ma il suo unico vero difetto è il formato ridotto della carta: appena 80 mm. di larghezza!
La terza macchina esaminata è la Epson P40, prodotta in due versioni, collegabili ai più comuni tipi d'interfaccia per stampante: la P40S è un modello seriale dotato di una presa RS232C, mentre la P40P possiede sul retro una porta parallela Centronics. Poiché la versione standard dello Spectrum manca di entrambe queste porte, è necessario acquistare l'Interface 1 oppure una delle altre numerose interfacce disponibili sul mercato.
Purtroppo la porta RS232 dell'Interface 1 è un connettore D a 7 prese non-standard, ma i manuali per la P40 e per l'Interface 1 descrivono come collegare e saldare i piedini di collegamento. In alternativa, si può acquistare una delle interfacce prodotte da altre ditte. La Kempston, ad esempio, fabbrica interfacce innestabili sul retro dello Spectrum e connettori standard per porte RS232 e Centronics.
Nonostante le ridotte dimensioni, la Epson 40 è capace di produrre stampati su 40 o su 80 colonne e risponde a un gran numero di codici Escape, abitualmente disponibili solo su apparecchi di maggiori dimensioni. Poiché lo Spectrum non possiede il tasto Escape, i codici vengono trasmessi alla stampante come codici ASCII usando il formato CHR$(27): CHR$(27);"E", ad esempio, ordina alla P40 di selezionare la stampa "in neretto".
La P40 offre anche altre possibilità, quali la selezione di set di caratteri alternativi, l'abilitazione del "bit mode" (che permette di creare nuovi caratteri) e il modo "condensato". Oltre a queste modifiche, accessibili via software, l'apparecchio possiede alcuni interruttori tipo DIP, per la selezione diretta della parità e del numero standard di colonne di stampa.
Il prestigioso marchio Epson garantisce che le stampanti P40 potranno in futuro contare su una regolare fornitura di materiali. Purtroppo questi apparecchi sono piuttosto cari e il loro prezzo, sommato a quello di un'interfaccia, supera il costo dello stesso computer.
A differenza delle precedenti stampanti, la Brother HR-5 stampa su carta termica oppure, purché dotata di nastro, anche su carta normale. Inoltre il suo carrello può accogliere fogli di formato A4, adatti alla stesura di lettere e ad altre applicazioni standard di word processing. Grazie a queste dimensioni "maggiorate", la Brother può stampare fino a 132 caratteri per linea.
Come gli apparecchi Epson, la HR-5 usa in abbondanza i codici Escape per formattare i testi e, inoltre, può produrre una vasta gamma di caratteri, una serie di corpi tipografici internazionali e vari formati di stampa. Il suo maggior pregio è la silenziosità ma, come i modelli Epson, possiede un'interfaccia parallela Centronics o una porta seriale RS232: occorre quindi acquistare un'interfaccia adeguata per il collegamento allo Spectrum. A parte ciò, la qualità della stampa dell'HR-5 è eccellente.
Dovendo scegliere una stampante, si rammenti dunque che in genere sono gli apparecchi con capacità superiori rispetto a quelli progettati appositamente per lo Spectrum che possono provocare inconvenienti: l'Alphacom e il Floyd 40 si collegano direttamente al computer e accettano i normali comandi dello Spectrum, come COPY, tuttavia le loro capacità si limitano alla stampa dei listati.
D'altro canto le stampanti di maggior prezzo, che offrono un maggior numero di applicazioni non sono progettate specificamente per lo Spectrum e possono presentare problemi di compatibilità insormontabili.

Controllo dinamico

Una serie per il sistema operativo CP/M della Digital Research



Quando comparvero sul mercato le prime memorie di massa a "floppy-disk", nacque l'esigenza di un sistema operativo capace di organizzare e gestire le informazioni immagazzinate sui dischetti. Durante questo periodo una compagnia americana produttrice di microchip, la Intel, decise di sviluppare un sistema a dischetti per i propri nuovi processori della serie 8000. Lo staff di ricerca della Intel, in quegli anni guidato da Gary Kildall, elaborò un programma chiamato CP/M (sigla inizialmente tratta da Control Program/Monitor e, in seguito, da Control Program for Microprocessor).

Questo nuovo sistema operativo fu lanciato nel 1975 e riscosse un folgorante successo. Alcuni dei suoi ideatori decisero di lasciare la Intel e fondare una propria compagnia, la Zilog Inc., e il primo chip fabbricato dalla nuova compagnia fu lo Z80, un processore compatibile con l'8080 e funzionante sotto CP/M. Lo stesso Kildall dette vita a una società, la Digital Research, oggi una delle più importanti del mondo.

Durante gli anni '70 il CP/M ha avuto molti concorrenti, poiché sono stati realizzati numerosi sistemi operativi per micro a 8 bit, soprattutto quelli basati sul chip 6502, incompatibile con i processori 8080 e Z80. Tuttavia il successo del CP/M ha del miracoloso se si considera che, anche a oltre 10 anni dalla sua comparsa, è ancora uno "standard" industriale per i micro gestionali a 8 bit!

Il CP/M può essere usato su computer dotati di un processore 8080 o compatibili, di un'unità a dischetti e di almeno 48 Kbyte di RAM. Questa quantità di memoria è necessaria per caricare il CP/M, i file di comando (che possono occupare fino a 8 Kbyte) e i file da elaborare e da registrare su dischetto.

L'avvento dei computer a 16 bit ha indotto molti a profetizzare un declino del CP/M nel mercato dei computer gestionali, ma il sistema continua a essere adottato da molti fabbricanti di home computer e di apparecchi didattici perché la sua flessibilità permette alle macchine basate sullo Z80 di abbandonare il mercato dei programmi di gioco per rivolgersi a quello delle applicazioni "professionali". Fornire un sistema operativo compatibile con il CP/M significa mettere a disposizione l'enorme quantità di pacchetti software sviluppati negli anni '70 per i computer basati su processori 8080 e Z80. Tra le compagnie che hanno recentemente imboccato questa strada vanno ricordate la Memotech, la Research Machines e l'Amstrad.

Mentre sulla maggior parte degli home computer il sistema operativo è "residente" (attivato al momento dell'accensione) il CP/M si trova su un dischetto (detto "disco di sistema") e viene caricato automaticamente quando entra in funzione l'unità a dischetti. Tuttavia, in alcuni casi i fabbricanti preferiscono dotare i propri apparecchi di un CP/M residente in ROM. Nel primo caso, la scelta ha motivazioni storiche: nei primi anni della rivoluzione informatica, la memoria era un "bene" prezioso e il CP/M residente in ROM avrebbe occupato memoria necessaria per altri scopi. Da qui la decisione di tenerlo su dischetto e di caricarlo solo quando era necessario.

Di solito il disco di sistema del CP/M contiene il programma del CP/M e una serie di piccoli programmi che possono essere richiamati (sempre tramite il CP/M) per abilitare specifiche funzioni di gestione del dischetto. Naturalmente il CP/M, oltre a gestire interamente il sistema d'archiviazione su disco, è un sistema operativo completo.

Quando il CP/M viene caricato su un computer dotato di un sistema operativo proprio su ROM, questo viene escluso, affidando il controllo dell'apparecchio al CP/M. Quando il computer gira sotto CP/M, i comandi BASIC normalmente impiegati dalla macchina non vengono riconosciuti dal nuovo sistema, che risponde loro con una serie di syntax error. Quindi i programmi in BASIC possono funzionare sotto CP/M soltanto se questo possiede un interprete o compilatore BASIC.

Sugli home computer, al momento dell'accensione, il CP/M viene caricato dal disco di sistema e sullo schermo compare l'elenco o "di-

rectory" dei comandi disponibili (vedere figura). Un esame della directory del dischetto fornisce numerose informazioni sulla struttura del CP/M. I nomi dei comandi (come del resto tutti quelli dei file in CP/M) si compongono di due parti, di cui la prima è il nome "primario". Per caricare un file di comando basta digitarne il nome seguito da quello del file sul quale si intende lavorare; il quale viene caricato ed elaborato automaticamente.
La seconda parte del nome (a destra del punto) è chiamata "estensione" e comunica al CP/M (e all'utente) la categoria alla quale appartiene il file. Poiché tutti i file del disco di sistema sono file di comando, la loro estensione è sempre COM. Gli altri tipi d'estensione dei file vengono esaminati in seguito.
Una particolarità della directory del disco è l'assenza del programma del CP/M. Ciò non significa che il CP/M sia un semplice "contenitore" di piccoli programmi di comando e non un programma a sé stante. Un'osservazione più approfondita della directory mostra che la lista contiene solo alcuni dei programmi indispensabili per il sistema (ad esempio, il comando LOAD) mentre altri (quali SAVE) non compaiono. Si è già detto che un buon SO deve essere il più "trasparente" possibile e ridurre al minimo le operazioni necessarie per dirigere l'intero sistema. Durante la fase di caricamento, il CP/M viene memorizzato in RAM, dove rimane finché il computer non viene spento, insieme a numerosi file di comando (detti "programmi di utilità") che così non devono essere ogni volta caricati dal disco. Per eliminare qualsiasi dubbio circa i file di comando residenti in RAM e quelli caricati di volta in volta, i primi sono chiamati "interni" e, come il programma del CP/M, non vengono visualizzati mentre i secondi, "transitori", compaiono in directory e, dopo la loro esecuzione, vengono cancellati dalla RAM e devono essere ricaricati per un successivo impiego.
Tale organizzazione, apparentemente arbitraria, dipende dalle restrizioni imposte dall'hardware e dal grado di flessibilità necessario per il corretto funzionamento del sistema. Poiché il CP/M è un SO basato su chip a 8 bit, il processore può indirizzare direttamente 64 Kbyte di memoria. Se tutti i comandi del CP/M venissero caricati all'accensione, la RAM a disposizione dell'utente non sarebbe sufficiente per contenere i programmi applicativi o i file di testo da elaborare. Ma ci sono altri vantaggi: per introdurre nuovi comandi transitori, infatti, è sufficiente creare programmi in codice macchina la cui estensione sia del tipo COM. Tale possibilità accresce la flessibilità del sistema e consente di utilizzare un'enorme varietà di programmi applicativi CP/M-compatibili. Ad esempio, il famoso pacchetto di word processing *WordStar*, che pure viene trattato come un programma separato eseguibile sotto CP/M, una volta caricato in memoria viene elaborato dal CP/M come se si trattasse di uno dei suoi file di comando. Grazie a ciò il *WordStar* può essere impiegato su qualsiasi apparecchio CP/M compatibile, poiché sfrutta le routine del CP/M per la gestione dell'input/output sia su disco sia sugli altri dispositivi, consentendo dimensioni ridotte dei programmi applicativi e un risparmio di memoria che può essere impiegata per immagazzinare, ad esempio, file di testo.
Nelle prossime parti dedicate al CP/M viene esaminato il funzionamento di alcuni programmi di utilità previsti dal sistema.

```
MBENMENU version 1.1
(c) 1984 Quantec Systems and Software Ltd
A)dir
A: MBENMENU COM : CPM3    SYS : COPYSYS  COM : ERASE   COM : SET     COM
A: SHOW     COM : SUBMIT  COM : TYPE     COM : DIR     COM : GET     COM
A: PUT      COM : RENAME  COM : SETDEF   COM : PIP     COM : HELP    COM
A: HELP     HLP : FC      COM : DEVICE   COM : FORMAT  SUB : FC      GET
A: FORMAT   GET : CONFIGUR COM
A)
```

La Directory del CP/M
Il comando DIR visualizza l'elenco dei file contenuti sul dischetto CP/M. Se l'utente non ha precedentemente salvato sul disco propri file, la directory del dischetto di sistema contiene soltanto i file di comando, riconoscibili dall'estensione .COM abbinata ai nomi di file, oltre a una serie di informazioni circa la memoria occupata da ciascun file







1981 1982 1983 1984 1985

Ratto baratto

Entriamo nel vivo delle trattative commerciali con gli abitanti del Nuovo Mondo

La linea 892 della sezione principale del programma chiama la routine baratto che, per prima cosa, verifica se il giocatore possiede fucili a bordo. Questi, indubbiamente utili per difendersi dai pirati o da altri nemici, sono tuttavia temuti dal capo indigeno, perché sa bene quali problemi possano creare tra la sua gente. Quindi, se il giocatore tenta di usare i fucili come merce di scambio, sullo schermo compare un messaggio di netto rifiuto da parte del capo. Ovviamente, se a bordo non esistono fucili, tale messaggio è del tutto inutile, perciò la linea 10072 esamina il secondo elemento OA(2) della matrice delle merci, relativo appunto ai fucili, e se questo è 0 non avviene la visualizzazione.

Le altre mercanzie, invece, vengono barattate in lotti separati.

La routine prosegue e comunica al giocatore qual'era il valore dei beni offerti dal capo indigeno, prima di partire per la traversata. Il primo prodotto trattato è il sale, che può essere barattato con perle, manufatti o spezie. Allo stesso modo vengono scambiate le stoffe, i coltelli e infine i monili.

La linea 10080 controlla se gli ultimi 4 elementi della matrice, da OA(3) a OA(6), contengono beni commerciabili. I primi due elementi non vengono esaminati poiché medicinali e fucili non interessano ai nativi. Per le altre mercanzie (coltelli, sale, stoffe e monili) il capo è disposto a offrire perle, manufatti e spezie. Se il giocatore ha poco senso del commercio (tutti e quattro gli elementi della matrice contengono il valore 0), il gioco termina perché non ci sono merci da barattare.

Dopo aver comunicato al giocatore il valore che le mercanzie avevano all'inizio del viaggio e avergli ricordato che il loro prezzo può essere cambiato nel frattempo, il programma entra in un ciclo (linee 10130-10200) che controlla il meccanismo del baratto. L'algoritmo calcola la quantità di perle, manufatti e spezie offerti dal capo per ciascun lotto di merce, invita il giocatore a scegliere una merce e la carica sulla nave.

Il contatore del ciclo assume i valori da 3 a 6 (ignorando medicinali e fucili) per trattare separatamente ciascun articolo. La linea 10135 controlla se l'elemento della matrice esaminato in quel momento contiene il valore 0: in tal caso il giocatore non possiede quel tipo di merce e il programma passa all'elemento successivo della matrice OA() saltando all'istruzione NEXT alla linea 10200. In caso contrario viene comunicata la quantità di merce disponibile (linea 10145). Le linee 10150-10153 visualizzano il tipo di merce che corrisponde al valore corrente di T.

Quando il capo ha fatto la sua offerta, il programma usa la matrice bidimensionale EQ(,), creata alla linea 63 e contenente i parametri di scambio, per calcolare la quantità di perle, manufatti o spezie che il giocatore riceve per ogni oggetto barattato.

L'espressione alla linea 10165 calcola la quantità di perle offerte per ogni merce, moltiplicando il numero di oggetti OA(T) che il giocatore intende scambiare per il parametro di scambio indicato nella matrice EQ(,). Poiché il conteggio del ciclo inizia da 3 e termina a 6, è necessario sottrarre 2 dal contatore T per ottenere il corretto indice da usare nella



matrice delle merci EQ(,) che va da 1 a 4. Se viene barattato il sale, T è uguale a 3; in questo caso EQ(T–2,1) corrisponde all'intersezione tra il primo elemento del primo indice, ossia il sale, e il primo elemento del secondo indice, che rappresenta le perle.
In EQ(,) tale intersezione vale 0.5, quindi la linea 10165 moltiplica per 0.5 la quantità di sale in modo da ottenere il numero di perle offerte in cambio.
La linea 10166 esegue un'operazione simile, ma sul numero di manufatti offerti dal capo indigeno. Il tasso di scambio per i manufatti è registrato nel secondo elemento del secondo indice di EQ(,).
Quindi i diversi valori assunti dal contatore del ciclo T indicano al programma la corrispondente sezione della seconda colonna della matrice.
Infine la linea 10167 usa la terza colonna della matrice dei tassi di scambio per calcolare la quantità di spezie offerte dal capo per ciascuna mercanzia del giocatore.

Accordo concluso

Dopo aver esaminato le offerte del capo, il giocatore deve comunicare la propria scelta, digitando 1, 2 o 3 a seconda del tipo di merce desiderato.
La linea 10176 verifica che il numero immesso sia compreso tra 1 e 3, mentre la linea 10180 registra la quantità di prodotti ricevuti in cambio nella matrice AO(), dimensionata alla linea 68. Infine le merci vengono caricate sulla nave.
I tre elementi della matrice AO() rappresentano i beni offerti dal capo. L'ammontare delle merci imbarcate viene calcolato moltiplicando l'appropriato elemento della matrice dei tassi di scambio EQ(,) per la quantità di beni barattati.
La variabile I, che contiene il numero digitato (ossia la scelta), ha una doppia funzione: viene usata dal programma sia come secondo indice di EQ(,), sia per selezionare il corrispondente elemento della matrice AO() nella quale vengono registrate le perle, i manufatti e le spezie ricevute in cambio.
Se in un successivo baratto il giocatore riceve una nuova partita di perle, manufatti o spezie, questa viene sommata al valore precedentemente registrato in AO() e il ciclo riprende dall'inizio. Quando tutte le merci sono state scambiate, la routine si conclude e un messaggio comunica al giocatore la quantità di perle, manufatti e spezie registrate nella matrice AO() e caricate sulla nave (linee 10220-10244). Infine il controllo torna al programma principale.
Nell'ultima parte di questo nostro progetto vengono presentate le routine che concludono il gioco.
Esse riguardano la possibilità di una rivolta tra gli indigeni, prima della partenza della nave, e la vendita delle merci imbarcate, nella speranza che i profitti siano tali da giustificare l'intera spedizione.

MODULO 12: BARATTO

Aggiunte al corpo principale del programma

```
892 GOSUB 10070
```

Routine Baratto

```
10070 PRINTCHR$(147): GOSUB 9200: REM BARATTO
10072 IF OA(2)=0 THEN 10080
10074 S$="IL CAPO RIFIUTA I FUCILI:•": GOSUB
      9100
10076 S$="TURBANO LA SUA GENTE!•": GOSUB 9100
10078 PRINT: GOSUB 9200
10080 IF OA(3)<>0 OR OA(4)<>0 OR OA(5)<>0 OR OA(6)<>0
      THEN 10100
10085 S$="UN INCENDIO HA DISTRUTTO•": GOSUB
      9100
10090 S$="TUTTE LE MERCI DA BARATTARE•": GOSUB
      9110
10095 GOTO 10038
10100 S$="IL CAPO INDIGENO TI OFFRE•": GOSUB
      9100
10102 S$="PERLE, MANUFATTI E SPEZIE•": GOSUB
      9100
10104 S$="IN CAMBIO DI COLTELLI,•": GOSUB
      9100
10106 S$="SALE, STOFFE E MONILI•": GOSUB 9100
10108 PRINT: GOSUB 9200
10110 S$="IL VALORE INIZIALE DELLE•": GOSUB
      9100
10112 S$="MERCI OFFERTE ERA:•": GOSUB 9100
10114 S$="1 PERLA = 2 MONETE D'ORO•":
      GOSUB 9100
10116 S$="1 MANUFATTO = 2 MONETE D'ORO•":
      GOSUB 9100
10118 S$="1 GR. SPEZIE = 1 MONETA D'ORO•":
      GOSUB 9100
10120 PRINT: GOSUB 9200
10122 S$="MA, RIMPATRIANDO, PUO'•": GOSUB
      9100
10124 S$="RISULTARE DIVERSO.•": GOSUB 9100
10125 PRINT: GOSUB 9200: S$=K$: GOSUB 9100
10126 GET I$: IF I$="" THEN 10126
10130 FOR T=3 TO 6
10135 IF OA(T)=0 THEN 10200
10140 PRINT CHR$(147): GOSUB 9200
10145 PRINT "ATTUALMENTE POSSIEDI";OA(T);
10150 IF T=3 THEN S$="SACCHI DI SALE•"
10151 IF T=4 THEN S$="BALLE DI STOFFA•"
10152 IF T=5 THEN S$="COLTELLI•"
10153 IF T=6 THEN S$="MONILI•"
10155 GOSUB 9200
10156 PRINT: GOSUB 9200
10160 S$="IL CAPO TI OFFRE:•": GOSUB 9100
10165 PRINT "       ";OA(T)•EQ(T-2,1);" PERLE"
10166 PRINT "     O ";OA(T)•EQ(T-2,2);"
      MANUFATTI"
10167 PRINT "OPPURE ";OA(T)•EQ(T-2,3);" GRAMMI DI
      SPEZIE•"
10168 PRINT: GOSUB 9200
10170 S$="DESIDERI PERLE, MANUFATTI•": GOSUB
      9100
10175 INPUT I$
10176 I=VAL(I$): IF I<1 OR I>3 THEN 10174
10180 AO(I)=AO(I)+(OA(T)•EQ(T-2,I))
10190 PRINT:PRINT "CARICO A BORDO DI ";T$(I);
      " EFFETTUATO."
10192 S$=K$: GOSUB 9100
10194 GET I$: IFI$="" THEN 10194
10200 NEXT
10210 PRINT: PRINT: GOSUB 9200
10215 S$="SCAMBIO DI MERCI TERMINATO•": GOSUB
      9100
10216 PRINT: GOSUB 9200
10218 S$="ADESSO POSSIEDI:•": GOSUB 9100
10220 PRINT AO(1);" PERLE"
10222 PRINT AO(2);" MANUFATTI"
10224 PRINT AO(3);" GRAMMI DI SPEZIE"
10226 PRINT: GOSUB 9200
10228 S$=K$: GOSUB 9100
10229 GET I$: IF I$="" THEN 10229
10230 RETURN
```

Basic & Basic

Spectrum:

Sostituire EQ(,) con Q(,) e AO() con E(). Apportare le seguenti modifiche:

```
10070 CLS
10126 LET I$ = INKEY$: IF I$ = "" THEN GO TO 10126
10140 CLS: GO SUB 9200
10229 LET I$ = INKEY$: IF I$ = "" THEN GO TO 10229
```

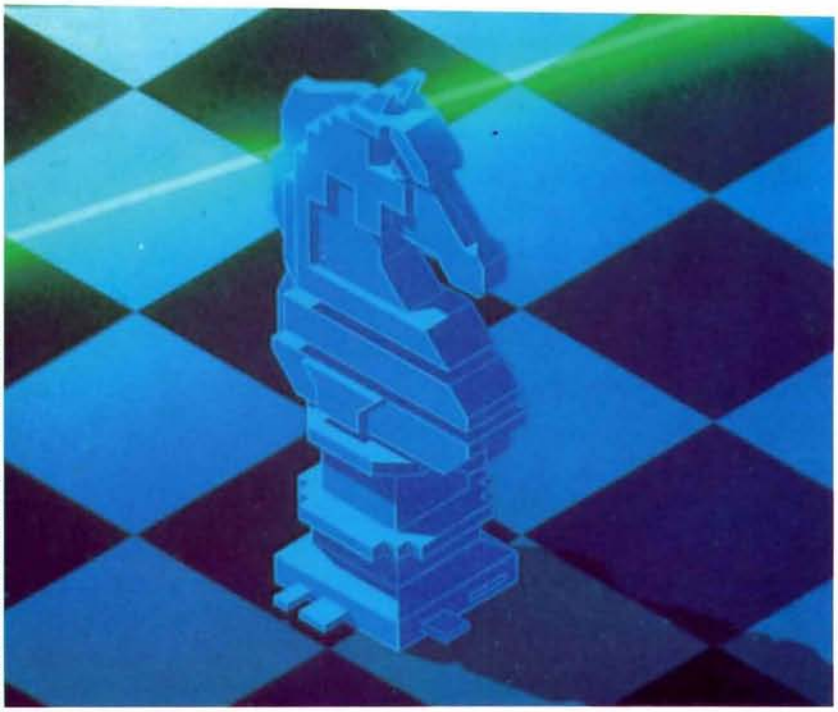
Micro BBC:

Apportare le seguenti modifiche:

```
10070 CLS
10126 I$ = GET$
10140 CLS: GOSUB 9200
```

Proiezioni strategiche

I meccanismi AI che consentono di prevedere le mosse di gioco



WATTLEWORTH SILK McCONACHIE con CHRIS TUCKER

Gran Maestro

La maggior parte dei programmi "intelligenti" per il gioco degli scacchi è in grado di prevedere un certo numero di mosse e di decidere qual'è la mossa successiva più conveniente. Per effettuare questa scelta, i computer costruiscono e analizzano un albero del gioco. Agli inizi della rivoluzione informatica, l'abilità di un computer nel gioco degli scacchi veniva considerata il più valido metro di valutazione per determinare l'"intelligenza" di un apparecchio

L'espressione inglese "computer game" evoca inevitabilmente immagini di astronavi aliene o di combattimenti tra gnomi in caverne sotterranee. Tuttavia, già nei primi anni della rivoluzione informatica, alcuni pionieri della scienza del computer, tra i quali Claude Shannon, John von Neumann e Alan Turing, si erano dedicati alla programmazione di computer capaci di giocare a scacchi.

Questo, infatti, era ritenuto uno tra i più impegnativi giochi d'intelletto e il corretto funzionamento di un programma per scacchi su computer veniva considerato un test straordinario per valutare le capacità (o l'intelligenza, qualcuno ha azzardato) di una macchina. Oggi esistono apparecchi come il Belle e il Cray Blitz che giocano a livelli paragonabili a quelli dei maestri internazionali, ma ciò non significa che queste macchine sappiano "pensare". Nonostante tutto, gli scacchi e altri giochi di destrezza mentale rappresentano un'arena ideale per verificare le teorie della proiezione strategica.

La maggior parte dei programmi di gioco si basa su tecniche di ricerca "ad albero", simili a quelle descritte a pag. 1721, anche se modificate per consentire la partecipazione di un avversario. Il principio al quale si ispirano è quello della "previsione". Il programma elabora una propria strategia di gioco, detta "albero del gioco", in base alla corrente disposizione dei pezzi sulla scacchiera, e opera la previsione di una serie di mosse in risposta a quelle dell'avversario.

Il diagramma dell'albero del gioco (vedere a pagina seguente) mostra l'albero di previsione per un'immaginaria partita a "filetto" tra due giocatori. L'origine o "radice" dell'albero rappresenta la situazione corrente, con MAX pronto a muovere. I nodi terminali, o foglie, indicano le possibili posizioni di fine-partita. L'albero viene usato in una particolare procedura (che nella teoria dei giochi è detta "minimax"), usata per selezionare la mossa più conveniente. La tecnica minimax, formulata per la prima volta in modo completo da Claude Shannon nel 1949, assegna un valore numerico ai nodi terminali. Ad esempio: 1 per una vittoria, 0 per un pareggio e −1 per una sconfitta. Via via che l'albero si sviluppa, e supponendo che il giocatore (MAX) scelga sempre il valore più alto e l'avversario (MIN) quello più basso, tali valori si combinano per produrre i valori dei nodi di livello superiore.

Nel nostro esempio, il valore della radice è 0 e ciò significa che la partita si conclude con un pareggio (a patto che nessun giocatore commetta errori). Al primo livello, la mossa giusta è M1, M3 o M4, ma non M2. Le regole che governano le biforcazioni e la generazione dei valori per i nodi variano da gioco a gioco. Solo in quelli più semplici, come "filetto", è possibile prevedere esattamente l'intero albero del gioco. Gli scacchi, ad esempio, possiedono un "fattore di biforcazione" di 32: perciò, per ogni disposizione dei pezzi, esistono approssimativamente 32 mosse consentite. Poiché una semplice previsione a 4 livelli (due mosse per ogni giocatore) produce oltre un milione di nodi terminali, i programmi di scacchi non sono in grado di prevedere lo sviluppo dell'intera partita.

In pratica, quasi tutti i package di scacchi prevedono il maggior numero possibile di situazioni e scelgono una mossa basandosi sull'ultima previsione. Occorre quindi un metodo capace di valutare se e in che misura un nodo terminale sia favorevole o no, anche se non è possibile ottenere una risposta valida in assoluto. Questa procedura viene chiamata "funzione di valutazione statica" ed è relativamente imprecisa, dal momento che si tratta semplicemente di una stima approssimata del probabile risultato finale. Tuttavia anche un'indagine di questo tipo, basata su una funzione di valutazione imperfetta, permette di ottenere previsioni attendibili sullo sviluppo e la conclusione di una partita.

Nel gioco della Dama o degli Scacchi, ad esempio, è possibile ideare una semplice funzione di valutazione basata sui 4 seguenti parametri:

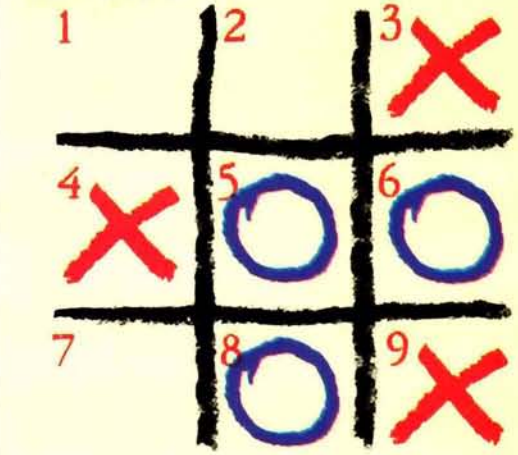
R vantaggio di Re
P vantaggio di Pezzo
M diversità di Mobilità
C controllo del Centro della scacchiera

Questi vengono via via calcolati dal programma esaminando la scacchiera. Il vantaggio di Re, per esempio, può essere determinato con l'espressione R=WR-BR, dove WR e BR indicano rispettivamente il numero di pezzi che difendono e che attaccano il Re. Le altre caratteristiche rispecchiano situazioni diverse: conviene avere più pezzi dell'avversario, poter disporre di un maggior numero di mosse e controllare le caselle centrali (più importanti di quelle ai bordi della scacchiera). In base a questi parametri il programma deve formulare una strategia di gioco vincente.
Supponendo che il Re (R) valga tre volte un pedone (P), che "mangiare" un pezzo equivalga a un vantaggio di 2 mosse e mezzo (M) e che una mossa in più abbia lo stesso valore del controllo di una casella centrale, si ottiene la seguente funzione di valutazione:

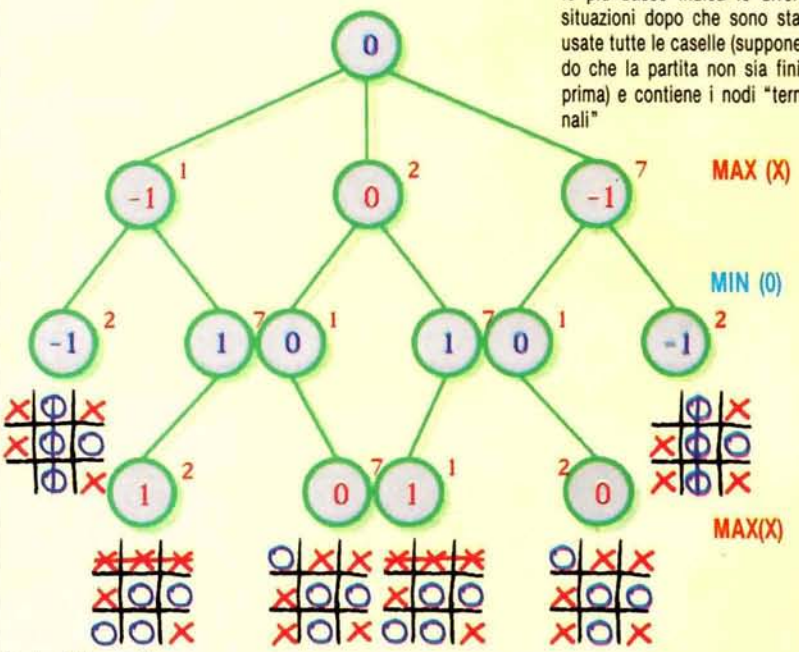
$$V = 15R + 5P + 2M + C$$

Per velocizzare il calcolo, spesso vengono usati numeri interi per i valori "ponderati" da abbinare ai parametri.
La funzione di valutazione descritta, però, si dimostra approssimativa e i coefficienti di valutazione arbitrari se si confronta tutto ciò con il programma di scacchi sviluppato all'inizio degli anni '60 da Arthur Samuel, che impiegava ben 25 parametri. Parte del divertimento, nella progettazione dei programmi di scacchi, è la "calibratura" dei valori ponderati assegnati ai parametri per ottenere una migliore valutazione della situazione di gioco. Uno dei principali pregi del programma di Samuel era la modifica automatica dei valori ponderati (qualcosa di simile a una rudimentale funzione di apprendimento).
L'idea di assegnare valori numerici alle diverse situazioni di gioco e di combinare tali valori in una somma ponderata, per valutare il valore di ciascuna posizione, ha dimostrato tutta la sua importanza negli anni successivi al programma di Samuel. La funzione di valutazione gioca un ruolo simile a quello della misurazione euristica della distanza impiegata nella soluzione dei problemi mediante ricerca, descritta a pag. 1721.
Un programma capace di prevedere solo un numero fisso di mosse e di valutare i nodi terminali ottenuti incontrerà grosse difficoltà, perché alcune posizioni di gioco sono "sicure" ma altre risultano "instabili". Negli scacchi, la situazione del gioco in seguito alla cattura di un pezzo o all'avanzamento di un pedone è instabile, in quanto può essere seguita dall'immediata cattura di un altro pezzo. Se questa avviene a un livello successivo all'ultimo previsto dal programma, la valutazione della mossa può dimostrarsi errata.
Per ridurre un tale rischio, la maggior parte dei programmi non esegue previsioni di profondità prefissata, ma si affida a una misurazione detta "di quiescenza", che indica il grado di affidabilità nella valutazione di una posizione. Se la posizione non può essere valutata con certezza, la ricerca prosegue con un'analisi delle mosse successive. Nei programmi di scacchi ciò implica la previsione di lunghe sequenze di cattura.
L'algoritmo "alfa-beta", usato nel pacchetto *MacHack* di Greenblatt è uno sviluppo della tecnica "MINIMAX" e produce gli stessi risultati ma con minor sforzo. Il diagramma a pagina seguente mostra parte dell'albero di previsione di una partita tra due giocatori (MINI e MAX). Le lettere assegnate a ciascun nodo (da A a L) indicano l'ordine nel quale viene esaminato l'albero da una procedura "depth-first" (con precedenza all'esame in profondità). I numeri rappresentano le valutazioni. La barretta singola rappresenta il cosiddetto "taglio alfa", la doppia il "taglio beta" e servono per eliminare le diramazioni che non influiscono sul risultato finale.

CHI VINCERÀ?



Il diagramma mostra la situazione, dopo 6 mosse, di una partita "filetto" tra i due giocatori "croce" e "zero" (la mossa spetta a "croce"). Basandosi sul numero di scelte consentite in ciascuna fase, è possibile costruire un semplice albero del gioco per prevedere i tre turni finali. Il primo livello contiene tre nodi corrispondenti alle tre mosse a disposizione di "croce" (nelle caselle 1, 2 e 3). Quando è il turno di "zero", sono rimaste vuote solo due caselle ed esistono dunque soltanto due scelte possibili delle tre disponibili prima che muovesse "croce". Il livello successivo contiene quindi 6 nodi. Il livello più basso indica le diverse situazioni dopo che sono state usate tutte le caselle (supponendo che la partita non sia finita prima) e contiene i nodi "terminali"



Scelta della casella

Dopo aver costruito l'albero del gioco, si attribuisce un valore a ogni nodo terminale: 1 per la vittoria di croce, 0 per un pareggio e —1 per la vittoria di zero. A questo punto si può risalire l'albero, assegnando valori agli altri nodi. Al nodo più a destra del primo livello viene assegnato il valore —1, in base ai suoi due nodi discendenti che hanno il valore di 0 e —1. Poiché l'albero riflette la strategia di "zero" (è il suo turno di gioco) viene selezionato il valore minimo, cioè —1. Risalendo alla prima mossa, "croce" dovrebbe scegliere il valore massimo dei 3 disponibili. Nell'esempio, la mossa più conveniente per "croce" è tracciare il proprio segno sulla casella 2, perché le altre due scelte farebbero vincere l'avversario

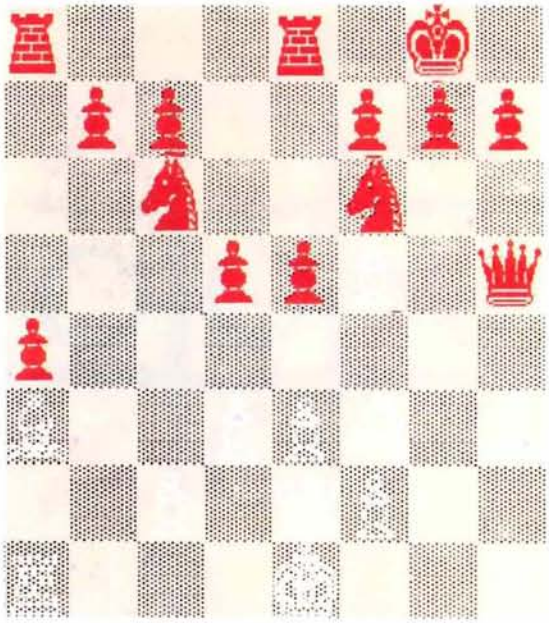
LIZ DIXON

Situazione della partita dopo 21 mosse

Questa è una della serie di partite che il miglior programma di scacchi per computer del mondo, il Cray Blitz, giocò e perse contro David Levy della Intelligent Software. Benché il programma abbia ricevuto riconoscimenti lusinghieri, è ancora necessario molto lavoro prima che i computer possano rappresentare un avversario veramente imbattibile per i migliori giocatori umani di scacchi

MOSSE:		
1	d2-d3	e7-e5
2	Nb1-d2	Nb8-c6
3	g2-g3	Ng8-f6
4	Bf1-g2	Bf8-c5
5	e2-e3	d7-d5
6	Ng1-e2	0-0
7	a2-a3	Bc8-f5
8	b2-b3	Nf6-g4?!
9	h2-h3	ng4-f6
10	Bc1-b2	Qd8-d6
11	g3-g4	Bf5-e6
12	Ne2-g3	a7-a5
13	Qd1-e2	Nf6-d7
14	Ng3-f5	Be6xf5
15	g4xf5	Nd7-f6
16	h3-h4	Rf8-e8
17	Bg2-h3	a5-a4
18	b3-b4	Bc5xb4!?
19	a3xb4	Qd6xb4
20	Bb2-a3	Qb4xh4
21	Nd2-f3	Qh4-h5
22	Nf3-d2	Qh5xe2+?
23	Ke1xe2	b7-b5
24	c2-c3	h7-h6
25	Bh3-g2	Ra8-a5
26	Ra1-b1	Re8-b8
27	Rb1-b2	Rb8-b6
28	Rh1-b1	Kg8-h7
29	Ba3-c5	Rb6-b8
30	Bc5-a3	Nc6-a7
31	Nd2-f3	Nf6-d7
32	Nf3-e1	c7-c6
33	Nd1-c2	Rb8-a8
34	Nc2-b4	Ra8-d8
35	Nb4-a2	Na7-c8
36	c3-c4!	d5xc4
37	d3xc4	b5xc4
38	Bg2xc6	Nc8-a7
39	Bc6-e4	Nd7-f6
40	Ba3-e7	Rd8-c8
41	Be7xf6	g7xf6
42	Rb1-b6!	c4-c3
43	Rb6xf6	Kh7-g7
44	Rf6-b6	a4-a3
45	Rb1-g1+	

A questo punto il programma ha comunicato di trovarsi in svantaggio di 2,4 pedoni e i programmatori hanno deciso di abbandonare



Un taglio alfa precede il nodo E, che non verrà mai valutato né possiederà rami discendenti. Quando MAX raggiunge E, sa già che il nodo C procura un punteggio di 15, tuttavia la mossa dell'avversario può costringerlo a scegliere il nodo D che produce un punteggio di 10. In questo caso non esistono altre mosse da esaminare, sebbene questa strada sia evidentemente meno desiderabile di quella che conduce a C. Pertanto, gli altri discendenti di F possono essere trascurati. Lo stesso ragionamento, ma ribaltato, vale per il nodo I. Quando MAX lo raggiunge, sa che G procura un punteggio di 20. Il nodo H, con valore 25, è indubbiamente migliore ma, in questo caso, è MINI e non MAX a scegliere tra G e J e, ovviamente, opta per G. Non importa sapere, dunque, se I è più favorevole dato che MAX non potrà mai raggiungerlo.

Gli stessi concetti possono essere applicati a un albero genealogico: MAX è un maschilista e pensa che il nodo C, ad esempio, sia lo zio dei nodi D e E che sono figli dello stesso padre. MINI, invece, è una femminista e per quanto la riguarda, G è la zia delle sorelle H e I, entrambe figlie di J, la madre. Se non ci si lascia ingannare dal cambiamento di sesso dei nodi a livelli alternati, questa analogia permette di chiarire in modo soddisfacente le regole dell'algoritmo alfa-beta:

- Non appena MAX trova che un figlio è "peggiore" di uno dei suoi zii, ignora la presenza degli altri fratelli.
- Non appena MINI trova che una figlia è "migliore" di una delle sue zie, ignora la presenza delle altre sorelle della figlia.

Nel migliore dei casi, rispetto alla tecnica minimax, l'algoritmo alfa-beta esamina due volte, e non una, la radice quadrata del numero dei nodi terminali dell'albero del gioco. Nel peggiore, li esamina tutti, ma in un tempo un po' superiore. Per evitare il primo dei due casi, è necessario generare fratelli e sorelle in ciascun livello seguendo un ordine ragionato. Ai livelli di massimizzazione, questi devono essere generati a partire dal migliore, a quelli di minimizzazione a partire dal peggiore (il migliore per l'avversario).

Il gioco dei numeri

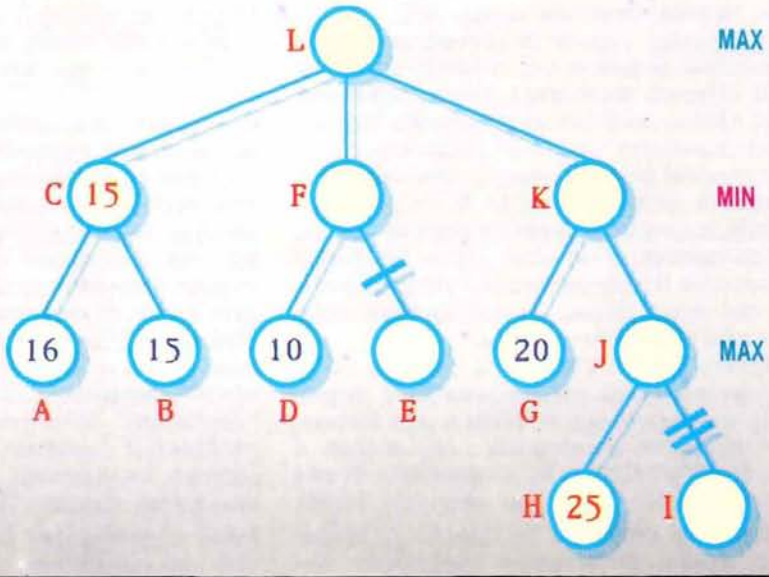
Per chiarire i principi fondamentali della ricerca su alberi, è utile esaminare un ipotetico gioco basato su tecniche di ricerca "pure", ossia privo di procedure quali la visualizzazione della scacchiera, la generazione di movimenti e la valutazione statica (essenziali per il funzionamento di un programma di gioco reale e diverse per ogni tipo di gioco), che potrebbero mascherare la semplicità di base della procedura alfa-beta.

Mancando sia la tastiera sia i pezzi, la situazione del gioco dipende interamente da un solo valore numerico contenuto in V%. L'obiettivo del computer-giocatore è quello di far crescere il valore di V% fino a 255, mentre il suo avversario cerca di portarlo a −255.

Ogni giocatore, al proprio turno, può applicare una delle quattro funzioni programmate alle linee 1030-1060. Queste possono essere

"POTATURA" DELL'ALBERO

La procedura alfa-beta elimina i rami inutili dell'albero, ottimizzando il funzionamento del metodo di base "minimax". I tagli alfa (indicati da una barretta singola) vengono eseguiti dopo che è stata individuata una strada minima per la mossa di MINI e non occorrono ulteriori ricerche. Viceversa, i tagli beta (rappresentati da una doppia barretta) vengono introdotti dopo l'individuazione di una strada massima per la mossa di MAX



LIZ DIXON

modificate, per creare altre versioni del gioco (ad esempio più difficili per il computer).
Il gioco è piuttosto elementare, ma permette di evidenziare la strategia alla base della ricerca e, basandosi esclusivamente su funzioni matematiche, offre un leggero vantaggio al computer. In questa versione, la procedura "minimax" poggia essenzialmente sull'uso di funzioni ricursive in cui sono usate variabili locali e i seguenti parametri:

VV% Situazione corrente del gioco
A% Il "migliore" di alfa fino al livello attuale
B% Il "peggiore" di beta fino al livello attuale
D% Indicatore di "profondità" della previsione

Poiché il programma prevede un processo ricursivo, viene presentata per prima la versione in BASIC del BBC. Le versioni non ricursive, per Commodore 64 e Spectrum, sono descritte in un secondo momento.

IL GIOCO DEI NUMERI

```
 10 REM ******************************
 11 REM **  LISTATO 3.1 :           **
 12 REM ** IL GIOCO DEI NUMERI      **
 13 REM ******************************
 50 MODE 7
100 REM -- Gioco dimostrativo per la ricerca:
120 GOSUB 1000  : REM inizializzazione
130 GOSUB 1600  : REM istruzioni
150 REPEAT
160   GOSUB 2000  : REM prepara una nuova partita
170   INPUT "Chi gioca per primo ? (1=Tu, 2=Io), H1%
180   IF H1%<1 OR H1%>2 THEN GOTO 170
200   REM -- ciclo principale:
210   REPEAT
220     IF H1%=1 THEN GOSUB 3000
230     REM -- turno del giocatore
240     GOSUB 3500  : REM visualizza lo schermo di gioco
250     H1%=1  : REM sempre 1 dopo il primo ciclo
260     GOSUB 4000  : REM decreta il vincitore
270     IF EG%=0 THEN GOSUB 5000
280     REM -- turno del computer.
290     GOSUB 3500: REM visualizza la situazione del gioco
300     GOSUB 4000: REM verifica se la partita e' terminata
310     UNTIL EG%<>0 OR M%>33
320   REM -- conclusione:
330   GOSUB 6000: REM CONGRATULAZIONI
340   PRINT "Un'altra partita ? (N=No) ";
350   Y$=GET$
360   UNTIL Y$="N" OR Y$="n"
365 PRINT
370 PRINT "Grazie per la bella partita!"
400 END
444 :
500 DEF FNmovmaxi (VV%, A%, B%, D%)
505 REM --------------- turno del computer:
510 LOCAL P%,E%,KEEP%
515 IF D%>=MD% OR ABS (VV%)>HI% THEN =VV%: REM valore statico
520 REM altrimenti il numero cala ancora:
525 P%=0
530 REPEAT P%=P%+1
535   H%=P%: V%=VV%: GOSUB 5500: REM muove
536   IF D%=1 THEN PRINT CHR$(H%+64);" = ";
540   E%=FNmovmini (V%, A%, B%, D%+1)
545   IF E%>A% THEN A%=E%: KEEP%=P%
548   IF D%=1 THEN PRINT E% ;"; ";
550   UNTIL P%>3 OR A%>=B%
555 IF A%>BV% AND D%>=1 THEN BV%=A%: H%=KEEP%
556 REM sceglie il migliore fino ad ora.
560 =A%
565 REM ritorna con valore max A%
570 :
700 DEF FNmovmini (VV%, A%, B%, D%)
710 REM ---------------- turno del giocatore :
720 LOCAL E%,P%
730 IF D%>MD% OR ABS(VV%)>HI% THEN =VV%
740 P%=0
750 REPEAT P%=P%+1: H%=P%
755   V%=VV%: GOSUB 5500: REM muove
760   E%=FNmovmaxi (V%, A%, B%, D%+1)
770   IF E%<B% THENB%=E%
780   UNTIL B%>3 OR B%<A%
790 =B%
796 REM  -- ritorna con valore min B%
999 :
1000 REM -- routine d'inizializzazione:
1001 BL$=""
1002 @%=4
1020 REM -- le 4 funzioni:
1030 DEF FNA(X%)=2*X% - 7
1040 DEF FNB(X%)=X% DIV 2 + 1
1050 DEF FNC(X%)=-4*X% + 17
1060 DEF FND(X%)=3*X% - 4
1070 LO%=-255: HI%=255
1150 RETURN
1160
1600 REM -- routine d'istruzione:
1610 CLS: PRINT
1620 PRINT "Benvenuto al Gioco dei Numeri!"
1630 PRINT "Se non conosci le regole,"
1635 PRINT "LEGGI LE ISTRUZIONI!"
1636 PRINT "N.B. Io massimizzo, tu minimizzi."
1640 PRINT "Per vedere il risultato di una mossa, usa"
1645 PRINT "A, B, C, o D. Premi X prima di ogni mossa."
1650 PRINT : PRINT "Buona Fortuna!"; CHR$(7)
1660 RETURN
1670
2000 REM -- routine di preparazione:
2010 M%=0: REM mosse
2020 V% = RND(15)-8: REM  situazione iniziale.
2050 EG%=0
2060 PRINT' "Situazione iniziale = ";V%
2100 RETURN
2110
3000 REM -- mossa del giocatore
3010 M%=M%+1
3020 PRINT
3030 REPEAT
3040   PRINT "La tua mossa e' ? ";
3050   H$=GET$: PRINT H$;
3060   IF H$="A" THEN PRINT FNA(V%): H%=1
3070   IF H$="B" THEN PRINT FNB(V%): H%=2
3080   IF H$="C" THEN PRINT FNC(V%): H%=3
3090   IF H$="D" THEN PRINT FND(V%): H%=4
3100   UNTIL H$="X"
3120 GOSUB 5500: REM esegue la mossa H%
3150 RETURN
3160 :
3500 REM -- routine di visualizzazione:
3520 CLS: PRINT
3522 PRINT "     Mossa ";M%;" -->  ";
3523 IF M%>1 THEN RETURN
3525 PRINT CHR$(64+H%);
3530 PRINT " = "; V%
3535 RETURN
3700
4000 REM -- routine per il vincitore (su M$):
4001 IF M%<1 THEN RETURN
4010 EG%=0
4020 IF V%<LO% THEN EG%=-1
4030 IF V%>HI% THEN EG%=1
4040 RETURN
4050 :
5000 REM -- routine per la mossa del computer:
5005 W%=V%: REM salva il punteggio corrente.
5010 M%=M%+1
5015 MD%=6: REM profondita' max della previsione
5020 IF M%<4 THEN MD%=4
5030 IF M%>8 THEN MD%=8
5040 GOSUB 5200: REM --> H%
5045 V%=W%: REM ripristina la situazione.
5050 GOSUB 5500: REM esegue la mossa.
5070 RETURN
5080 :
5200 REM -- scelta della mossa:
5210 BV%=LO%: D%=0
5220 BV%=FNmovmaxi (V%,LO%,HI%,1)
5230 H%=HH%
5240 PRINT' "(un tasto qualsiasi per continuare): ";
5244 C%=GET
5250 RETURN
5270 :
5500 REM -- routine di esecuzione della mossa (H% : V%):
5505 ON H% GOTO 5510,5520,5530,5540
5510 V%=FNA(V%): RETURN
5520 V%=FNB(V%): RETURN
5530 V%=FNC(V%): RETURN
5540 V%=FND(V%): RETURN
5550 :
6000 REM -- routine delle Congratulazioni:
6010 PRINT "FINE DELLA PARTITA!"
6020 IF EG>0 THEN PRINT "Hai perso!!"
6030 IF EG<0 THEN PRINT "Complimenti, hai vinto!"
6040 IF EG=0 THEN PRINT "Pareggio!"
6050 RETURN
6600 :
```

Un prologo sul Prolog

La prima parte di una serie dedicata a questo linguaggio

L'industria giapponese è stata la prima ad impegnarsi nella progettazione di computer della cosiddetta "Quinta Generazione". Come linguaggio di base per questi nuovi apparecchi, dalle elevate prestazioni e dotati di database "intelligenti", fu scelto il PROLOG che, specialmente agli inizi, è rimasto sconosciuto al grande pubblico.

La sigla PROLOG è un acronimo derivato dalla definizione inglese "programming in logic", ossia "programmazione della logica" ed è a tale principio che il linguaggio aderisce, nonostante qualche limitazione naturale. Occorre esaminare, dunque, quali siano i vantaggi di una programmazione completamente "logica". Esistono vari tipi di logica in base alla quale è possibile descrivere il mondo che ci circonda; alcuni, come la matematica, sono piuttosto familiari, ma altri sono noti soltanto a una ristretta cerchia di studiosi (è il caso di alcune dottrine filosofiche). Il "calcolo predicato di primo ordine" è una logica abbastanza simile adoperata per articolare i normali pensieri di una persona, tuttavia è caratterizzato da una propria notazione e da particolari restrizioni. Nel linguaggio corrente, un predicato indica una relazione tra due "argomenti". Nella frase "A Paolo piace Anna", ad esempio, il predicato è "piace". Lo stesso concetto può essere espresso nella forma seguente: "piace (Paolo, Anna)", che è meno immediata ma meglio chiarisce quale sia il predicato e quali i suoi argomenti. Per indicare che Paolo è un maschio, si può scrivere "maschio(Paolo)" dove "maschio" è un predicato riferito a un argomento (Paolo). Ugualmente, "femmina(Anna)" significa che Anna appartiene al sesso opposto.

Le precedenti proposizioni non sono che semplici dichiarazioni di un fatto, ma il principio della logica predicata può essere esteso per chiarire come alcuni fatti ne "implichino" altri. Si può cioè descrivere la realtà sotto forma di "fatti" e "implicazioni" e usare una propria rappresentazione per dedurre nuovi fatti da quelli noti. Una simile operazione implica l'introduzione di variabili logiche, concettualmente simili a quelle comunemente usate in BASIC o in altri linguaggi di programmazione, eccetto che per il loro scopo, che è limitato alla "clausola" (fatto o implicazione) nella quale compaiono. In altre parole, il Paolo al quale piace Anna può essere un Paolo diverso da quello prima associato alla definizione "maschio".

Esaminiamo adesso una proposizione (fatto) del tipo: "femmina (X) → piace (Paolo,X)". Poiché la freccia significa "implica", la frase può essere interpretata nel modo seguente: "il fatto che X sia femmina implica che a Paolo piace X", ossia "A Paolo piace X se X è femmina". Questa regola d'interpretazione è un esempio di un particolare tipo di clausola del calcolo predicato, chiamata "horn clause" (clausola a corni). Le horn clause presentano una dichiarazione di testa (detta "conseguenza") che è vera solo se tutte le istruzioni del corpo della proposizione (chiamate "antecedenti") sono vere:

A è "vero" se B e C e D sono tutti "veri"

Questa è una horn clause composta dalla testa A e dal corpo B, C e D. Quindi un singolo fatto può essere interpretato come una conseguenza priva di antecedenti e considerato vero.

Programmazione nella logica

La logica predicata permette di scrivere un programma come sequenza di fatti e regole che rappresentano l'argomento da elaborare, senza discostarsi troppo dalla forma nella quale il problema è stato pensato. Per ottenere un risultato da un programma logico, va dimostrata la veridicità o la falsità di alcune dichiarazioni. Se la dichiarazione è un fatto singolo, lo si può considerare vero senza ulteriori interventi. Se invece è la conseguenza di alcune regole, occorre provare la veridicità di tutti i suoi antecedenti prima di dichiararlo vero. Volendo sapere se Paolo è maschio, si deve cercare di dimostrare la dichiarazione "maschio(Paolo)", che è considerata "vera" in quanto data per scontata. Se invece vogliamo sapere se Anna piace a Paolo, vanno prima verificate le dichiarazioni "maschio(Paolo)" e "piace(Paolo, Anna)".

Il PROLOG è stato sviluppato agli inizi degli anni '70 da A. Colmerauer, un docente dell'Università di Marsiglia, che si è in parte rifatto al lavoro di Bob Kowalsky. Egli usò tre sole horn clause della logica predicata e una notazione simile a quella descritta in queste pagine. Sarebbe stato possibi-



LIZ HEANEY

PROLOG per Micro

Per funzionare al meglio, il PROLOG deve poter disporre di una notevole quantità di RAM: per questa ragione le versioni del linguaggio destinate ai microcomputer sono piuttosto rare. Tuttavia chi possiede uno Spectrum può acquistare il MICRO-PROLOG, una versione del linguaggio su cassetta prodotta dalla Logic Programming Associates e distribuita dalla Sinclair.

Ma anche gli utenti di altri micro hanno buone speranze poiché il Prolog ha sollecitato l'attenzione dei produttori di software, che si sono immediatamente cimentati nella creazione di versioni adatte al Micro BBC e all'Enterprise.

Nella serie dedicata a questo linguaggio, gli esempi di programmazione sono redatti nella versione standard (DEC-10 PROLOG), che per molti aspetti differisce dal MICRO-PROLOG. Tuttavia, come al solito, le varianti vengono evidenziate a parte, per consentire l'immissione dei listati usando anche il MICRO-PROLOG.

le implementare l'intero calcolo predicato ma, come altri linguaggi di programmazione, il PROLOG è un compromesso tra efficienza dell'elaborazione e facilità di comprensione. Sebbene l'attuale struttura del PROLOG sia ormai consolidata, si stanno ancora eseguendo ricerche per affinarne il formato.

La versione di PROLOG che più si avvicina a uno standard è la DEC-10 (così chiamata perché è stata sviluppata su un grosso elaboratore della Digital Equipment Corporation), mentre la "bibbia" dei programmatori in PROLOG è il libro scritto da Clocksin e Mellish e intitolato *Programming in PROLOG*. Questo contiene una descrizione dell'ultima versione del linguaggio e numerose informazioni pratiche. Quasi tutti i dialetti PROLOG oggi in commercio, compreso il C-PROLOG, sono modellati su questo standard. Per i micro sono disponibili due versioni: quella della Expert Systems, molto vicina alla versione standard, e il MICRO-PROLOG della Logic Programming Associates, che ha riscosso un buon successo ma presenta notevoli differenze di sintassi e di struttura interna.

Le versioni estese del PROLOG occupano una discreta quantità di memoria e creano problemi di caricamento sui micro con meno di 64 Kbyte di RAM. Tuttavia esiste un MICRO-PROLOG per lo Spectrum e ogni giorno nascono nuove versioni di questo dialetto.

I programmi PROLOG non seguono l'abituale flusso di controllo sequenziale (dopo la prima istruzione del programma viene eseguita la seconda e così via fino all'ultima con l'interpolazione di diramazioni e cicli iterativi), ma adottano una tecnica di ricerca inversa chiamata "backtracking".

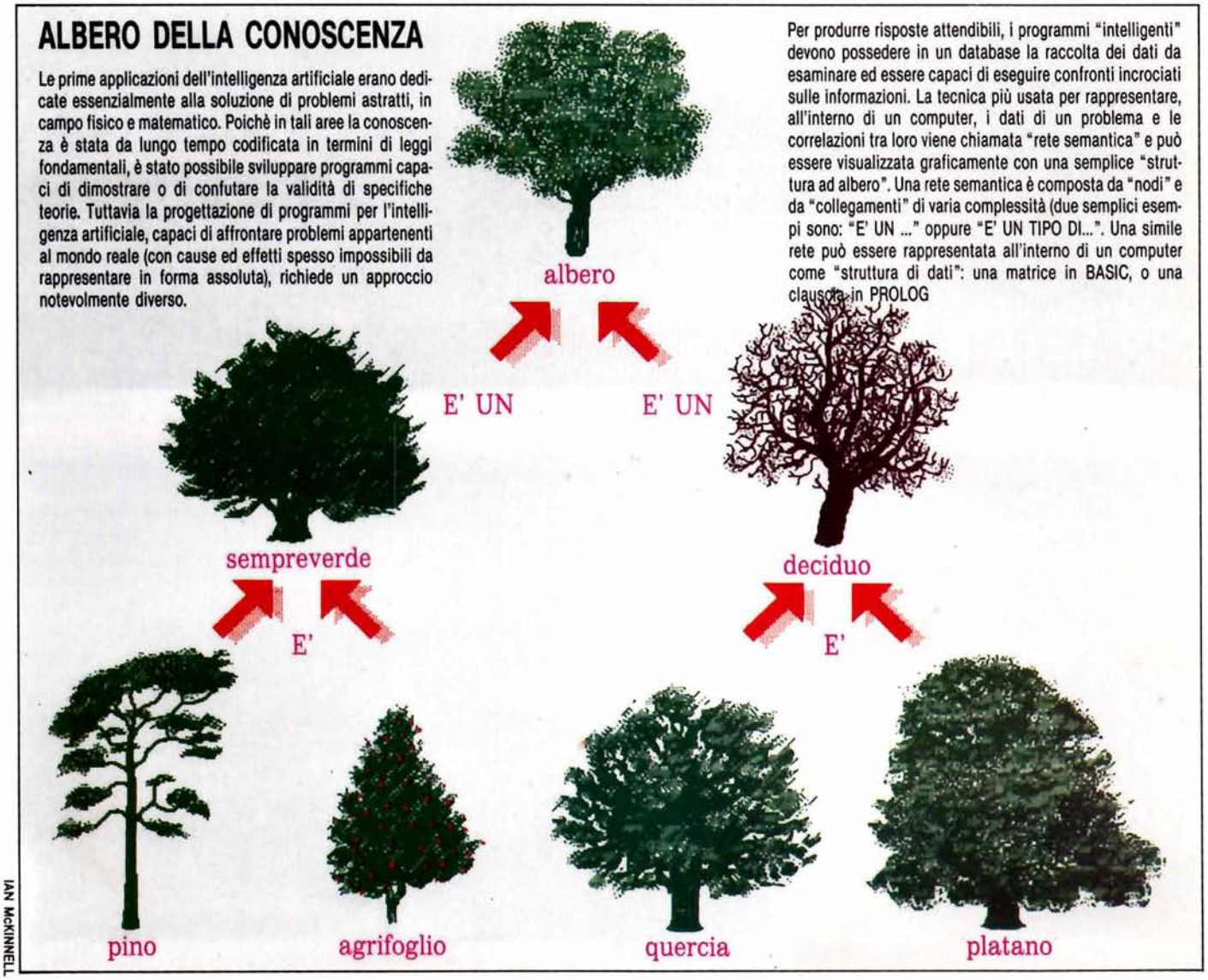
Per rispondere a un quesito, il PROLOG cerca la soluzione in base a una serie di regole concatenate, ponendosi di volta in volta un nuovo obiettivo da raggiungere. Se una particolare sequenza di elaborazione si dimostra inefficace, il PROLOG torna indietro all'ultima scelta fatta e imbocca una nuova direzione. Una dettagliata analisi dell'argomento viene presentata più avanti.

Questo originale modo di pocedere differenzia il PROLOG da tutti gli altri linguaggi di programmazione. Inoltre, i fautori della programmazione logica sottolineano la natura dichiarativa dei programmi PROLOG, ossia la possibilità offerta dal calcolo predicato di primo ordine di "interpretare" o "leggere" una regola come se si trattasse di una clausola. Ad esempio: X è zio di Y se X è maschio e X è il fratello di Z e Z è il padre di Y. In più il PROLOG può sempre essere interpretato nello stile procedurale più lineare (per dimostrare che X è zio di Y, per prima cosa si può provare che X è maschio, quindi che X è fratello di Z e infine che Z è il padre di Y). L'interpretazione dichiarativa, caratteristica assente in quasi tutti gli altri linguaggi, favorisce la comprensione dei problemi e aiuta il programmatore a progettare e collaudare i propri programmi.

ALBERO DELLA CONOSCENZA

Le prime applicazioni dell'intelligenza artificiale erano dedicate essenzialmente alla soluzione di problemi astratti, in campo fisico e matematico. Poichè in tali aree la conoscenza è stata da lungo tempo codificata in termini di leggi fondamentali, è stato possibile sviluppare programmi capaci di dimostrare o di confutare la validità di specifiche teorie. Tuttavia la progettazione di programmi per l'intelligenza artificiale, capaci di affrontare problemi appartenenti al mondo reale (con cause ed effetti spesso impossibili da rappresentare in forma assoluta), richiede un approccio notevolmente diverso.

Per produrre risposte attendibili, i programmi "intelligenti" devono possedere in un database la raccolta dei dati da esaminare ed essere capaci di eseguire confronti incrociati sulle informazioni. La tecnica più usata per rappresentare, all'interno di un computer, i dati di un problema e le correlazioni tra loro viene chiamata "rete semantica" e può essere visualizzata graficamente con una semplice "struttura ad albero". Una rete semantica è composta da "nodi" e da "collegamenti" di varia complessità (due semplici esempi sono: "E' UN ..." oppure "E' UN TIPO DI..."). Una simile rete può essere rappresentata all'interno di un computer come "struttura di dati": una matrice in BASIC, o una clausola in PROLOG



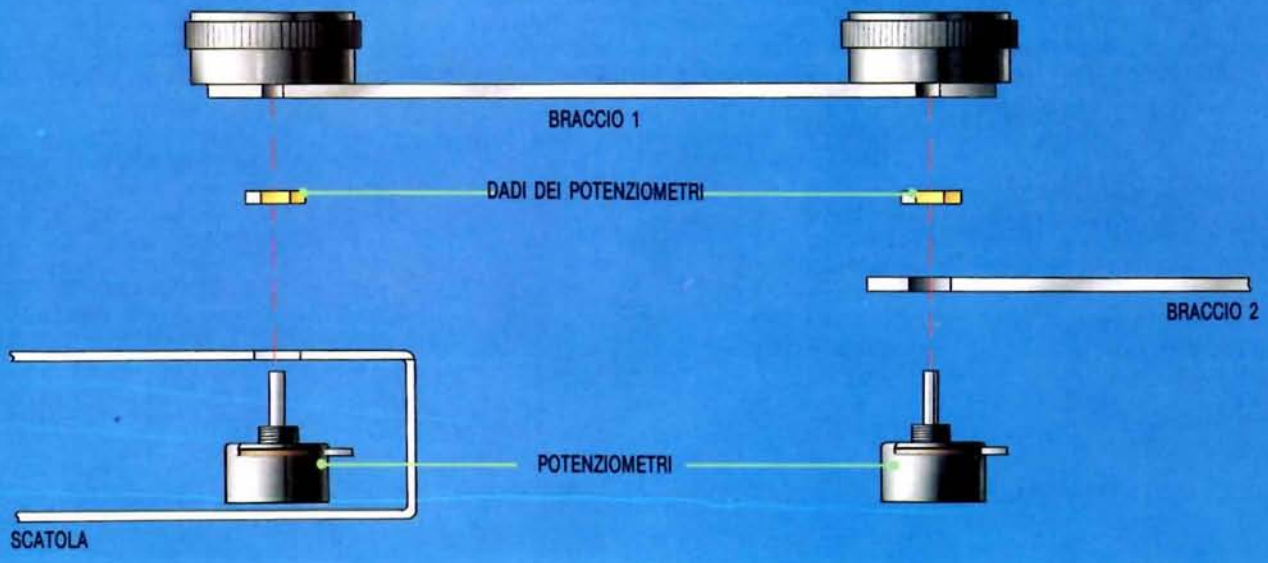
IAN McKINNELL

Montaggio del tracer

Prima di passare al software di controllo, va completata la costruzione dell'apparecchio

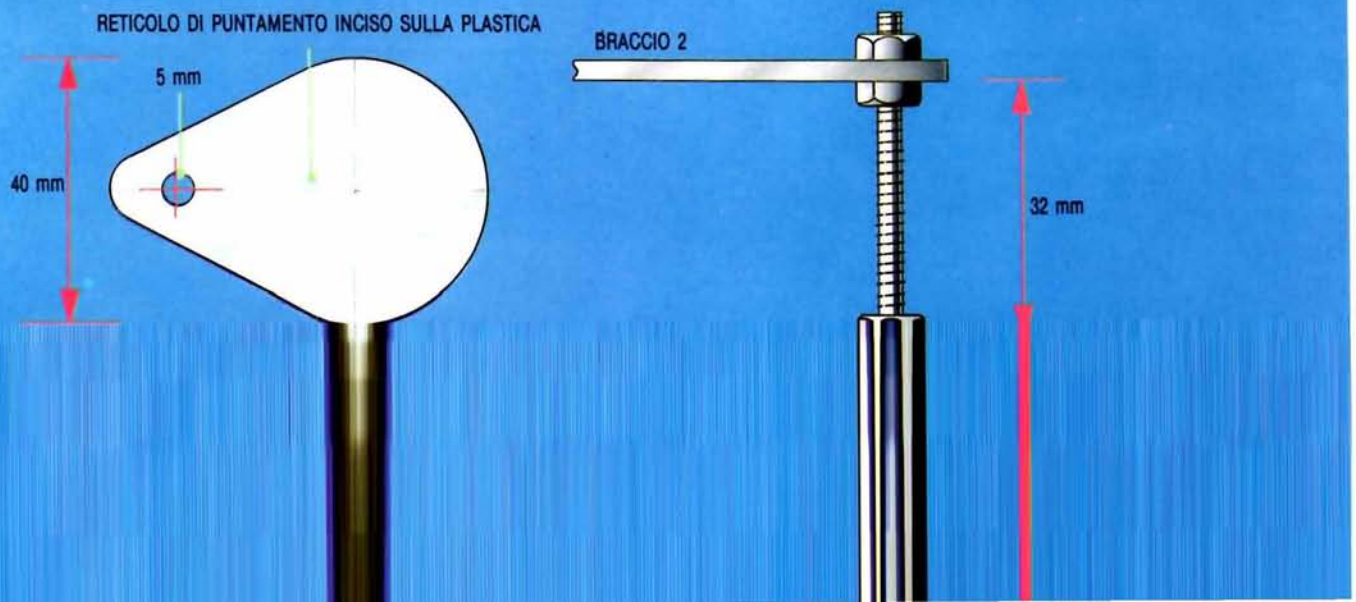
FASE 1: ASSEMBLAGGIO DEL BRACCIO

I due segmenti del braccio, realizzati in precedenza, fanno perno su una coppia di potenziometri: uno viene montato sul coperchio della scatola di plastica fissata alla base dell'apparecchio, mentre l'altro collega tra loro i due segmenti. Gli alberini devono essere tagliati a una distanza di 15 mm. dal corpo dei potenziometri. Questi vengono poi fissati al coperchio della scatola e al braccio 2, bloccandoli con gli appositi dadi. Le linguette di collegamento del potenziometro situato nella scatola devono essere rivolte verso la parete più vicina, mentre quelle dell'altro vanno allineate al braccio (vedere figura). Gli alberini vengono quindi inseriti nelle rispettive manopole, già montate sul braccio 1, e fissati stringendo le viti di bloccaggio. Se gli alberini hanno una superficie piatta, questa va rivolta verso la vite di bloccaggio.

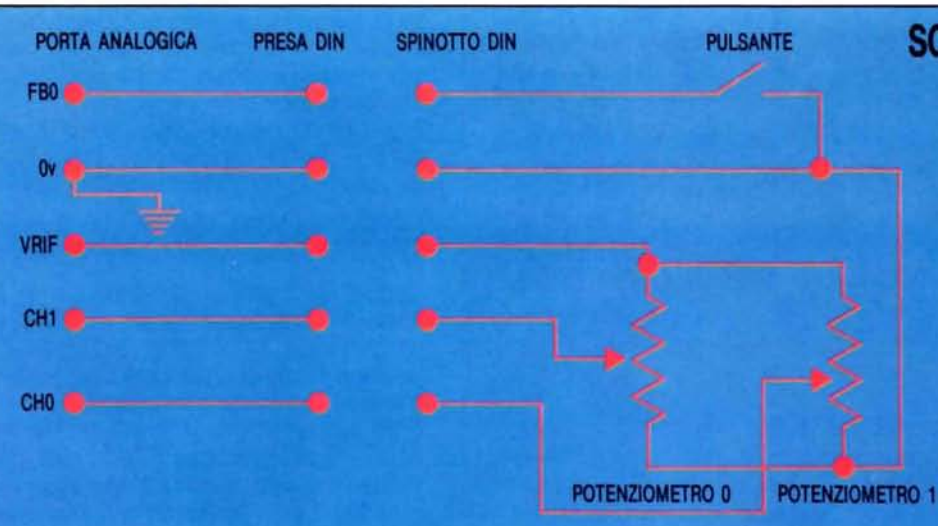


FASE 2: MONTAGGIO DEL MIRINO

Il mirino o traguardo si realizza adoperando una lamina di plastica trasparente (può essere usata la parte trasparente di una custodia per musicassette). Le dimensioni non sono critiche, ma le linee del reticolo di puntamento, che va inciso sulla superficie, devono incrociarsi a 27 mm. di distanza dal centro del foro di montaggio. Il puntatore viene fissato all'estremità libera del Braccio 2 usando un bulloncino e tre dadi. Questi permettono di modificare a piacere la sua distanza dal braccio che, normalmente, dovrebbe essere di circa 32 mm.



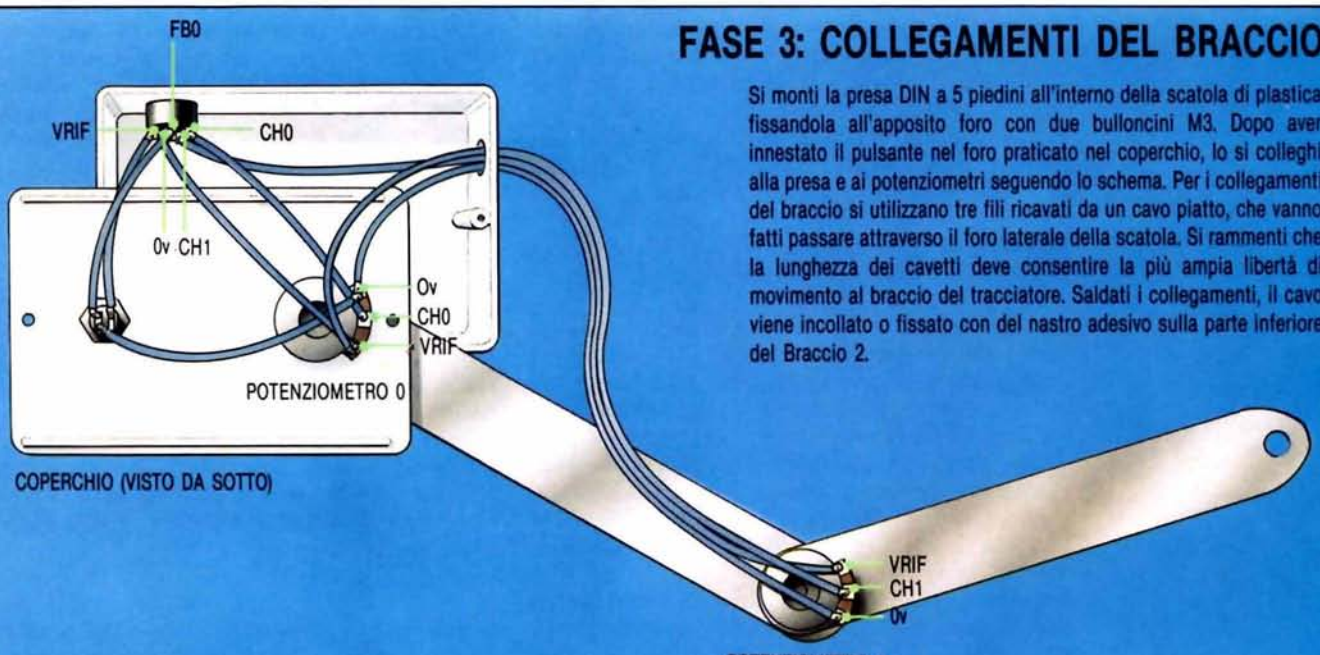
SCHEMA DEL CIRCUITO



Il diagramma mostra il circuito elettrico del tracciatore digitale. Il pulsante normalmente aperto è collegato alla linea FB0 della porta analogica, che corrisponde al "pulsante di fuoco" di un joystick. I segnali vengono decodificati in seguito dal software. La tensione di riferimento usata nella conversione analogico/digitale viene inviata alle due piste del potenziometro: i collegamenti centrali dei potenziometri forniscono i segnali di ritorno, con i canali 0 e 1, al convertitore A/D, che li trasforma in segnali digitali. Questa conversione consente di calcolare le angolazioni del braccio e la posizione del puntatore del "tracciatore".

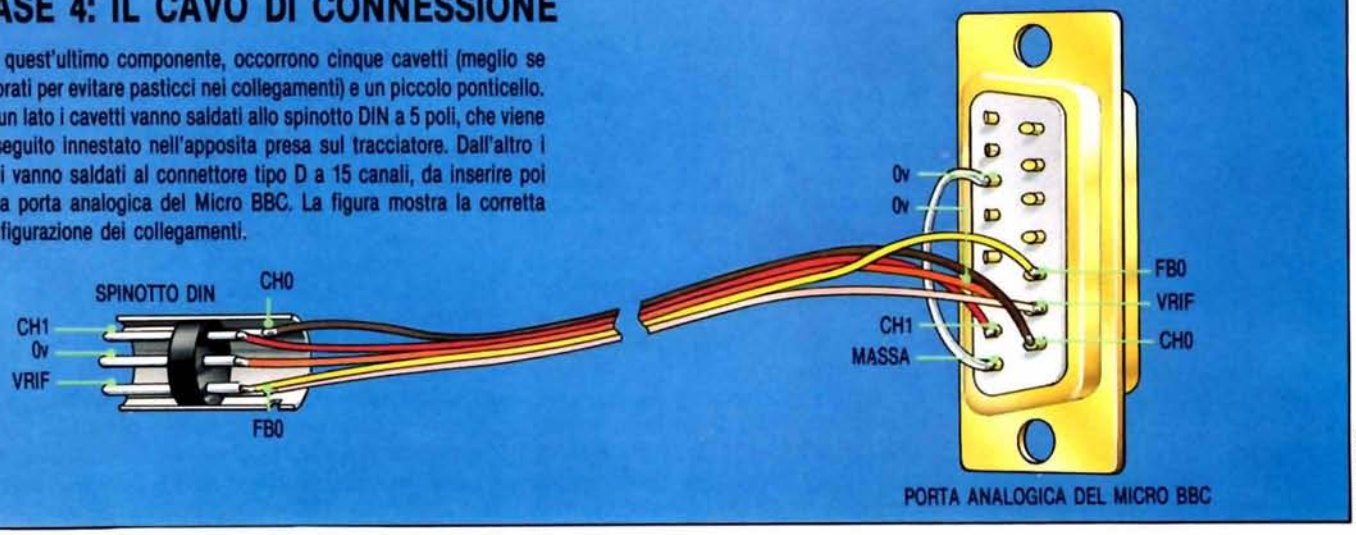
FASE 3: COLLEGAMENTI DEL BRACCIO

Si monti la presa DIN a 5 piedini all'interno della scatola di plastica fissandola all'apposito foro con due bulloncini M3. Dopo aver innestato il pulsante nel foro praticato nel coperchio, lo si colleghi alla presa e ai potenziometri seguendo lo schema. Per i collegamenti del braccio si utilizzano tre fili ricavati da un cavo piatto, che vanno fatti passare attraverso il foro laterale della scatola. Si rammenti che la lunghezza dei cavetti deve consentire la più ampia libertà di movimento al braccio del tracciatore. Saldati i collegamenti, il cavo viene incollato o fissato con del nastro adesivo sulla parte inferiore del Braccio 2.



FASE 4: IL CAVO DI CONNESSIONE

Per quest'ultimo componente, occorrono cinque cavetti (meglio se colorati per evitare pasticci nei collegamenti) e un piccolo ponticello. Da un lato i cavetti vanno saldati allo spinotto DIN a 5 poli, che viene in seguito innestato nell'apposita presa sul tracciatore. Dall'altro i capi vanno saldati al connettore tipo D a 15 canali, da inserire poi nella porta analogica del Micro BBC. La figura mostra la corretta configurazione dei collegamenti.



KEVIN JONES

Rivoluzione grafica

La seconda e ultima parte dedicata alla grafica 3-D sul Commodore 64

Il programma ibrido (*Test I-Rot e I-Rot.Hex*), presentato a pag. 1736, funziona abbastanza velocemente. Tuttavia la scansione della matrice E%(I,J), necessaria per individuare i punti che devono essere tracciati, rallenta l'esecuzione del programma. La matrice E%(,), si ricorderà, definisce i nodi collegati tra loro da linee visibili. Per eliminare questo inconveniente occorre programmare in codice macchina il resto del ciclo BASIC del programma *Cubo Rotante* (a pag. 1739) e inviare in esecuzione il ciclo in codice macchina. Solo così si ottiene un aumento di velocità.
Per poter eseguire i seguenti calcoli:

```
X1% = X(I) + 159: Y1% = 199—(Z(I) + 100)
X2% = X(J) + 159: Y2% = 199—(Z(J) + 100)
```

originariamente inseriti nelle linee 1650 e 1660 del programma BASIC, è necessario chiamare delle nuove routine dell'interprete. In concreto, entrambe le linee sommano il valore 159 (espresso in virgola mobile) alla variabile in virgola mobile X(I), prima di prenderne la parte intera e immagazzinarla, come variabile intera X1%, in 2 byte. Ecco le routine dell'interprete che fanno il lavoro:

- FLPINT (indirizzo $B1AA):

Questa routine inserisce in FAC la parte intera del numero, calcola il risultato (se compreso tra i valori —32767 e 32767) nel formato byte-basso/byte-alto e lo trascrive rispettivamente nei registri Y e A. Si noti l'insolita disposizione basso/alto, opposta a quella della maggior parte delle routine dell'interprete.

- SNGFT (indirizzo $B3A2):

Questa routine estrae un intero a singolo byte (compreso tra 0 e 255) dal registro Y e lo inserisce in FAC nel formato a virgola mobile.
SNGFT viene usata dalla subroutine SETUP del listato Assembly (linea 5150) che inserisce il valore 159 nel registro Y e chiama la routine SNGFT per trasformare il numero e inserirlo in FAC. Quindi usa MOVMF per copiare il risultato nei 5 byte di MEM1, dove viene conservato e successivamente impiegato quando occorre sommare 159.
Gli altri problemi relativi alla conversione in codice macchina del ciclo BASIC riguardano il caricamento degli elementi della matrice che definisce la forma della figura da ruotare. Infatti, in alcuni casi, il calcolo dei puntatori alla matrice può risultare difficoltoso. D'altra parte le matrici X(I), Y(I), Z(I) delle coordinate non creano particolari problemi, poiché basta aggiungere 5 byte al puntatore in ciascuna matrice per ottenere l'indirizzo dell'elemento successivo.
La matrice bidimensionale E%(I,J) pone problemi diversi. I suoi elementi vengono registrati in memoria in modo sequenziale:

```
E%(0,0)E%(1,0)E%(2,0)...E%(NP,0)
E%(0,1)E%(1,1)E%(2,1)...E%(NP,1) e così via...
```

Quindi la matrice si compone di blocchi di memoria, ciascuno lungo 2x(NP+1): ogni blocco corrisponde ai valori del secondo indice e ciascun elemento occupa due byte (perché la matrice contiene valori interi).
Il nostro obiettivo è "tradurre" nel modo più esatto il programma BASIC nel codice macchina in modo che i cicli I,J, che esaminano la matrice E%(I,J), presentino il formato:

```
FOR I=1 TO NP
FOR J=1 TO I
```

Ciò complica notevolmente la modifica da apportare al puntatore, per ottenere un codice macchina equivalente a NEXT I, considerando che vanno saltati tutti gli elementi il cui primo indice è 0. Per ottenere la rotazione della figura, gli elementi di E%(I,J) vanno elaborati nel seguente ordine:

```
E%(1,1)
E%(1,2)E%(2,2)
E%(1,3)E%(2,3)E%(3,3)
E%(1,4)E%(2,4)E%(3,4)E%(4,4) ecc.
```

Un rapido calcolo dà l'espressione 2x(NP+1), ossia il valore da sommare al puntatore ogni volta che viene incrementato I. Usando il codice macchina del 6502, la tecnica più coveniente per accedere agli elementi di E%(I,J) è quella dell'indirizzamento indiretto che viene programmato con le istruzioni:

```
LDY JINDEX
LDA (ZPTEMP),Y
```

ZPTEMP è un puntatore a due byte situato in pagina zero, mentre JINDEX viene usato per registrare i valori di J. ZPTEMP va incrementato dopo ogni aumento di J. Dall'incremento applicato a ZPTEMP e al registro Y dipende l'aumento di due byte dell'offset necessario per ogni successivo incremento di J. Quindi, come risultato finale di queste complesse operazioni, ZPTEMP viene incrementato di (2xNP+1)—(I—1) ad ogni iterazione nel ciclo di I. Il valore (I—1) viene sottratto dalla lunghezza del blocco in quanto ZPTEMP è già stato incrementato (I—1) volte dal ciclo di J appena concluso. L'esecuzione di questa espressione per il calcolo dell'offset, porta il vettore ZPTEMP a puntare al giusto byte dopo che I è stato incrementato.
Un'ultima considerazione: sarebbe assai conveniente poter chiamare una routine dell'interprete capace di localizzare direttamente la variabile E%(I,J). Tale routine esiste ma sfortunatamente il suo uso è molto complesso (in effetti funziona con tutti i possibili tipi di variabile) e l'esecuzione lenta. L'offset, pertanto, va calcolato nel modo appena descritto, a partire dall'indirizzo di E%(1,1).

ROUTINE DI ROTAZIONE

Il seguente listato è relativo al programma in codice macchina che fa ruotare la figura geometrica definita dalla matrice E%(I,J). Il programma sfrutta le routine dell'interprete e le tecniche di individuazione delle variabili descritte nel testo. Il primo obiettivo (convertire in linguaggio macchina il ciclo BASIC che legge le matrici, per visualizzare una figura in rotazione) è dunque raggiunto. Inoltre vengono forniti un programma di caricamento in BASIC del codice macchina e un programma BASIC di prova che prepara le variabili per il codice macchina e chiama la routine. Questo programma può essere usato con una versione assemblata del codice sorgente ("II— ROT.HEX") o, in alternativa, caricando ed eseguendo il programma di caricamento, digitando il comando NEW e infine caricando e lanciando il programma di prova. Nel secondo caso si elimini la linea 1030 dal programma di prova

CARICATORE IN BASIC

```
1000 REM** INSERISCI LA VERSIONE II-ROT.HEX **
1010 REM*************************************
1020 DATA72,138,72,152,72,32,101,199
1030 DATA173,83,197,172,84,197,32,162
1040 DATA187,173,75,197,172,76,197,32
1050 DATA40,186,162,94,160,197,32,212
1060 DATA187,173,87,197,172,88,197,32
1070 DATA162,187,173,77,197,172,78,197
1080 DATA32,40,186,169,94,160,197,32,80
1090 DATA184,162,94,160,197,32,212,187
1100 DATA173,87,197,172,88,197,32,162
1110 DATA187,173,75,197,172,76,197,32
1120 DATA40,186,162,99,160,197,32,212
1130 DATA187,173,83,197,172,84,197,32
1140 DATA162,187,173,77,197,172,78,197
1150 DATA32,40,186,169,99,160,197,32
1160 DATA103,184,162,99,160,197,32,212
1170 DATA187,169,94,160,197,32,162,187
1180 DATA174,83,197,172,84,197,32,212
1190 DATA187,169,99,160,197,32,162,187
1200 DATA174,87,197,172,88,197,32,212
1210 DATA187,206,93,197,240,31,169,5,24
1220 DATA109,83,197,141,83,197,144,3
1230 DATA238,84,197,169,5,24,109,87,197
1240 DATA141,87,197,144,3,238,88,197,76
1250 DATA112,197,169,1,141,[illegible]193,141,1
1260 DATA193,141,2,193,32,14,173,32,101
1270 DATA199,169,1,141,81,197,141,82
1280 DATA197,173,79,197,133,253,173,80
1290 DATA197,133,254,172,82,197,177,253
1300 DATA200,[illegible],76,207,198,173,83,197
1310 DATA172,84,197,32,162,187,169,94
1320 DATA160,197,32,103,184,32,170,177
1330 DATA140,0,195,141,1,195,173,85,197
1340 DATA172,86,197,32,162,187,169,94
1350 DATA160,197,32,103,184,32,170,177
1360 DATA140,2,195,141,3,195,173,89,197
1370 DATA172,90,197,32,162,187,169,99
1380 DATA160,197,32,89,184,32,170,177
1390 DATA140,4,195,173,91,197,172,92
1400 DATA197,32,162,187,169,99,160,197
1410 DATA32,80,184,32,170,177,140,5,195
1420 DATA173,0,195,205,2,195,208,19,173
1430 DATA1,195,205,3,195,208,11,173,4
1440 DATA195,205,5,195,208,3,76,207,198
1450 DATA32,14,195,173,81,197,205,82
1460 DATA197,240,40,169,5,24,109,85,197
1470 DATA141,85,197,144,3,238,86,197
1480 DATA169,5,24,109,91,197,141,91,197
1490 DATA144,3,238,92,197,230,253,208,2
1500 DATA230,254,238,82,197,76,73,198
1510 DATA173,74,197,205,81,197,240,88
1520 DATA169,5,24,109,83,197,141,83,197
1530 DATA144,3,238,84,197,169,5,24,109
1540 DATA89,197,141,89,197,144,3,238,90
1560 DATA237,81,197,24,105,1,24,101,253
1570 DATA133,253,165,254,101,0,133,254
1580 DATA173,68,197,141,85,197,173,69
1590 DATA197,141,86,197,173,70,197,141
1600 DATA91,197,173,71,197,141,92,197
1610 DATA169,1,141,82,197,238,81,197,76
1620 DATA73,198,104,168,104,170,104,96
1630 DATA173,68,197,141,83,197,141,85
1640 DATA197,173,69,197,141,84,197,141
1650 DATA86,197,173,70,197,141,87,197
1660 DATA173,71,197,141,88,197,173,74
1670 DATA197,141,93,197,173,72,197,141
1680 DATA89,197,141,91,197,173,73,197
1690 DATA141,90,197,141,92,197,160,159
1700 DATA32,162,179,162,94,160,197,32
1710 DATA212,187,160,99,32,162,179,162
1720 DATA99,160,197,32,212,187,96
1730 DATA79160:REM*CHECKSUM*
1740 FOR I=50536 TO 51123
1750 READX: POKEI,X: CC=CC+X
1760 NEXT
1770 READX: IF X<>CC THEN PRINT "ERRORE
     CHECKSUM": STOP
1780 PRINT "II -ROT.HEX CARICAMENTO OK"
```

LISTATO ASSEMBLY

```
1010 ;++++++++++++++++++++
1020 ;++      ROTSUB 64     ++
1030 ;++                    ++
1040 ;++++++++++++++++++++++++
1140 ;
1150 *       = $C544
1190 ; VARIABILI CHIAMATE DAL BASIC
1200 ;
1210 XBASLO  *=*+1 ; POKE50500,X(1)LO
1220 XBASHI  *=*+1 ; POKE50501,X(1)HI
1230 XBASLO  *=*+1 ; POKE50502,Y(1)LO
1240 XBASHI  *=*+1 ; POKE50503,Y(1)HI
1250 XBASLO  *=*+1 ; POKE50504,Z(1)LO
1260 XBASHI  *=*+1 ; POKE50505,Z(1)HI
1270 NP      *=*+1 ; POKE50506,NP
1280 CSLO    *=*+1 ; POKE50507,CSLO
1290 CSHI    *=*+1 ; POKE50508,CSHI
1300 SNLO    *=*+1 ; POKE50509,SNLO
1310 SNHI    *=*+1 ; POKE505010,SNHI
1320 EBASLO  *=*+1 ; POKE505011,E%(1,1)LO
1330 EBASHI  *=*+1 ; POKE505012,E%(1,1)HI
1340 ;
1350 ; VARIABILI USATE DAL CODICE MACCHINA
1370 INDEX   *=*+1
1380 JNDEX   *=*+1
1390 XILO    *=*+1
1400 XIHI    *=*+1
1410 XJLO    *=*+1
1420 XJHI    *=*+1
1430 YILO    *=*+1
1440 YIHI    *=*+1
1450 ZILO    *=*+1
1460 ZIHI    *=*+1
1470 ZJLO    *=*+1
1480 ZJHI    *=*+1
1490 TEMPNP  *=*+1
1500 MEM1    *=*+5 ; VARIABILE A VIRGOLA MOBILE
1510 MEM2    *=*+5 ; VARIABILE A VIRGOLA MOBILE
1520 ZTEMP   =$FD  ; LOCAZIONE LIBERA DI PAG ZERO
1530 ;
1540 ; CHIAMATE ARITMETICHE DELL'INTERPRETE
1560 FMULT   =$BA28 ; FAC=FAC*MEM
1570 FSUB    =$B850 ; FAC=MEM-FAC
1580 FADD    =$B867 ; FAC=FAC+MEM
1590 MOVFM   =$BBA2 ; FAC=MEM
1600 MOVMF   =$BBD4 ; MEM=FAC
1610 FLPINT  =$B1AA ; .Y/.A=INT(FAC) N.B. FORMATO BASSO/ALTO!!
1620 SNGFT   =$B3A2 ; FAC= .Y
1630 ;
1640 ; ALTRE ROUTINE PER VARIABILI IN C/M
1660 LINSUB  =$C30E
1670 MLO     =$C300
1680 MHI     =$C301
1690 NLO     =$C302
1700 NHI     =$C303
1710 Y1      =$C304
1720 Y2      =$C305
1740 ;+++ SALVA I REGISTRI +++
1760         PHA
1770         TXA
1780         PHA
1790         TYA
1800         PHA
1820 ;+++ INIZIALIZZA LE VARIABILI +++
1840         JSR SETUP
1860 ;++++ CALCOLA MEM1=X(I)*CS-Y(I)*SN +++
1880 INIZIO
1890         LDA XILO
1900         LDY XIHI
1910         JSR MOVFM    ; FAC=X(I)
1920         LDA CSLO
1930         LDY CSHI
1940         JSR FMULT    ; FAC=X(I)*CS
1950         LDX #<MEM1
1960         LDY #>MEM1
1970         JSR MOVMF    ; MEM1=X(I)*CS
1980         LDA YILO
1990         LDY YIHI
2000         JSR MOVFM    ; FAC=Y(I)
2010         LDA SNLO
2020         LDY SNHI
2030         JSR FMULT    ; FAC=Y(I)*SN
```

```
2040        LDA #<MEM1
2050        LDY #>MEM1
2060        JSR FSUB     ; FAC=MEM1-FAC
2070        LDX #<MEM1
2080        LDY #>MEM1
2090        JSR MOVFM    ; MEM1=FAC
2110 ;+++ CALCOLA MEM2=Y(I)*CS+X(I)*SN +++
2130        LDA YILO
2140        LDY YIHI
2150        JSR MOVFM    ; FAC=Y(I)
2160        LDA CSLO
2170        LDY CSHI
2180        JSR FMULT    ; FAC=Y(I)*CS
2190        LDX #<MEM2
2200        LDY #>MEM2
2210        JSR MOVMF    ; MEM2=Y(I)*CS
2220        LDA XILO
2230        LDY XIHI
2240        JSR MOVFM    ; FAC=X(I)
2250        LDA SNLO
2260        LDY SNHI
2270        JSR FMULT    ; FAC=X(I)*SN
2280        LDA #<MEM2
2290        LDY #>MEM2
2300        JSR FADD     ; FAC=MEM2+FAC
2310        LDX #<MEM2
2320        LDY #>MEM2
2330        JSR MOVMF    ; MEM2=FAC
2350 ;+++ CALCOLA X(I)=MEM1:Y(I)=MEM2 +++
2370        LDA #<MEM1
2380        LDY #>MEM1
2390        JSR MOVFM    ; FAC=MEM1
2400        LDX XILO
2410        LDY XIHI
2420        JSR MOVMF    ; X(I)=FAC
2430        LDA #<MEM2
2440        LDY #>MEM2
2450        JSR MOVFM    ; FAC=MEM2
2460        LDX YILO
2470        LDY YIHI
2480        JSR MOVMF    ; Y(I)=FAC
2500 ;+++ VERIFICA DI FINE CICLO +++
2520        DEC TEMPNF
2530        BEQ CONTIN
2550 ;+++ INCREMENTA I PUNTATORI ALLA MATRICE +++
2570        LDA #$05
2580        CLC
2590        ADC XILO
2600        STA XILO
2610        BCC XNOHI
2620        INC XIHI
2630 XNOHI
2640        LDA #$05
2650        CLC
2660        ADC YILO
2670        STA YILO
2680        BCC YNOHI
2690        INC YIHI
2700 YNOHI
2710        JMP INIZIO
2730 ;+++ CANCELLA/INIZIALIZZA LO SCHERMO +++
2750 CONTIN
2760        LDA #$01
2770        STA $C100
2780        STA $C101
2790        STA $C102
2800        JSR $C10E    ; SELEZIONA L'ALTA RISOLUZIONE
2820 ;+++ TRACCIA LE LINEE DA E%(I,J) +++
2840 ; INIZIALIZZA LE VARIABILI
2860        JSR SETUP
2870        LDA #$01
2880        STA IINDEX   ; I=1
2890        STA JINDEX   ; J=1
2900        LDA EBASLO   ; MEMORIZZA IL
2910        STA ZPTEMP   ; PUNTATORE E%
2920        LDA EBASHI   ; IN PAGINA
2930        STA ZPTEMP+1 ; ZERO
2950 ; USCITA-INIZIA IL CICLO GIGANTE I,J
2970 SUCCIJ
2980        LDY JINDEX
2990        LDA (ZTEMP),Y ; GET E%(I,J)
3000        BNE ESEGUE
3010        JMP ONWARD
3030 ;+++ CALCOLA X1%=X(I)+159 +++
3040 ;    MLO=X1% LO:MHI=X1% HI
3060 ESEGUE
3070        LDA XILO
3080        LDY XIHI
3090        JSR MOVFM    ; FAC=X(I)
3100        LDA #<MEM1
3110        LDY #>MEM1
3120        JSR FADD     ; FAC=X(I)+159
3130        JSR FLPINT   ; .Y/.A=INT(FAC)
3140        STY MLO      ; X1% LO
3150        STA MHI      ; X1% HI
3170 ;+++ CALCOLA X2%=X(J)+159 +++
3180 ;    NLO=X2% LO:NHI=X2% HI
3200        LDA XJLO
3210        LDY XJHI
3220        JSR MOVFM    ; FAC=X(J)
3230        LDA #<MEM1
3240        LDY #>MEM1
3250        JSR FADD     ; FAC=X(J)+159
3260        JSR FLPINT   ; .Y/.A=INT(FAC)
3270        STY NLO      ; X2% LO
3280        STA NHI      ; X2% HI
3300 ;+++ CALCOLA Y1%=99-Z(I) +++
3320        LDA ZILO
3330        LDY ZIHI
3340        JSR MOVFM    ; FAC=Z(I)
3350        LDA #<MEM2
3360        LDY #>MEM2
3370        JSR FSUB     ; FAC=99-Z(I)
3380        JSR FLPINT   ; .Y/.A=INT(FAC)
3390        STY Y1       ; Y1%
3410 ;+++ CALCOLA Y2%=99-Z(J) +++
3430        LDA ZJLO
3440        LDY ZJHI
3450        JSR MOVFM    ; FAC=Z(J)
3460        LDA #<MEM2
3470        LDY #>MEM2
3480        JSR FSUB     ; FAC=99-Z(J)
3490        JSR FLPINT   ; .Y/.A=INT(FAC)
3500        STY Y2       ; Y2%
3520 ; PRONTO PER LINSUB
3540 ; CONFRONTO A DUE BYTE PER X1%=X2%
3560        LDA MLO
3570        CMP NLO
3580        BNE NOPE
3590        LDA MHI
3600        CMP NHI
3610        BNE NOPE
3630 ; CONFRONTA Y1=Y2
3650        LDA Y1
3660        CMP Y2
3670        BNE NOPE
3690 ; TUTTI UGUALI QUINDI EVITA LINESUB
3710        JMP ONWARD
3720 NOPE
3730        JSR LINSUB  ;TRACCIA UNA LINEA
3750 ; INCREMENTA I PUNTATORI A J
3770 ONWARD
3780        LDA IINDEX   ; PRIMA VERIFICA
3790        CMP JINDEX   ; J=<I ?
3800        BEQ NEXTJ
3820 ;INCREMENTA XJLO/XJHI
3840        LDA #$05
3850        CLC
3860        ADC XJLO
3870        STA XJLO
3880        BCC XJNOHI
3890        IN XIHI
3900 XJNOHI
3920 ; INCREMENTA ZJLO/ZJHI
3940        LDA #$05
3950        CLC
3960        ADC ZJLO
3970        STA ZJLO
3980        BCC ZJNOHI
3990        INC ZJHI
4000 ZJNOHI
4020 ; INCREMENTA DI UNO IL PUNTATORE A E%
4040        INC ZPTEMP
4050        BNE NOHIBY
4060        INC ZPTEMP+1
4070 NOHIBY
4080        INC JINDEX
4090        JMP NEXTIJ
4110 ; INCREMENTA I PUNTATORI
4130 SUCCI
4140        LDA NP       ; PRIMA VERIFICA
4150        CMP IINDEX   ; I=<NP ?
4160        BEQ EXIT
4180 ; INCREMENTA XILO/XIHI
4200        LDA #$05
4210        CLC
4220        ADC XILO
4230        STA XILO
4240        BCC XINOHI
4250        INC XIHI
4260 XINOHI
4280 ; INCREMENTA ZILO/ZIHI
4300        LDA #$05
4310        CLC
4320        ADC ZILO
4330        STA ZILO
4340        BCC ZINOHI
4350        INC ZIHI
4360 ZINOHI
4380 ; INCREMENTA I PUNTATORI A E%
4400        LDA #$04     ; QUANDO INCREMENTA I
4410        CLC          ; VA SOMMATO
4420        ADC NP       ; 4*(NP+1)-(I-1)
4430        ASL A        ; AL PUNTATORE
4440        SEC          ; ZP PER
4450        SBC IINDEX   ; E%
4460        CLC
4470        ADC #$01
4480        CLC
4490        ADC ZPTEMP
4500        STA ZPTEMP
4510        LDA ZPTEMP+1
```

```
4520        ADC #$00
4530        STA ZPTEMP+1
4550 ; REINIZIALIZZA XJLO/XJHI
4570        LDA XBASLO
4580        STA XJLO
4590        LDA XBASHI
4600        STA XJHI
4620 ; REINIZIALIZZA ZJLO/ZJHI
4640        LDA ZBASLO
4650        STA ZJLO
4660        LDA ZBASHI
4670        STA ZJHI
4690 ; REINIZIALIZZA JINDEX
4710        LDA #$01
4720        STA JINDEZ
4740 ; INCREMENTA IINDEX
4760        INC IINDEX
4770        JMP NEXTIJ
4790 ; TERMINA IL CICLO GIGANTE
4800 ;
4810 ;+++ ESTRAE I VALORI DAI REGISTRI +++
4820 ;
4830 USCITA
4840        PLA
4850        TAY
4860        PLA
4870        TAX
4880        PLA
4890        RTS
4930 ;+++ SUBROUTINE +++
4940 ;
4950 CREA
4960        LDA ZBASLO
4970        STA XILO
4980        STA XJLO
4990        LDA ZBASHI
5000        STA XIHI
5010        STA XJHI
5020        LDA YBASLO
5030        STA YILO
5040        LDA YBASHI
5050        STA YIHI
5060        LDA NP
5070        STA TEMPNP
5080        LDA ZBASLO
5090        STA ZILO
5100        STA ZJLO
5110        LDA ZBASHI
5120        STA ZIHI
5130        STA ZJHI
5140        LDA #159
5150        JSR SNGFT    ; FAC=159
5160        LDX #<MEM1
5170        LDY #>MEM1
5180        JSR MOVMF    ; MEM1=159 IN FPT
5190        LDA #99
5200        JSR SNGFT    ; FAC=99
5210        LDX #<MEM2
5220        LDY #>MEM2
5230        JSR MOVMF    ; MEM2=99 IN FTP
5240        RTS
```

PROGRAMMA DI PROVA BASIC

```
1000 REM** TRACCIA I CUBI IN ROTAZIONE **
1010 IF A=0 THEN A=1: LOAD"PLOTSUB.HEX",8,1
1020 IF A=1 THEN A=2: LOAD"LINESUB.HEX",8,1
1030 IF A=2 THEN A=3: LOAD"II -ROT.HEX",8,1
1040 PRINTCHR$(147) "ATTENDI 17 SECONDI"
1050 REM** DIMENSIONA LE MATRICI **
1060 NP=24: REM NUMERO DEI PUNTI
1070 DIM X(NP),Y(NP),Z(NP)
1080 DIM E%(NP,NP): REM COLLEGAMENTI DEGLI SPIGOLI
1090 REM** INIZIALIZZA LE MATRICI **
1100 REM -- COORDINATE INIZIALI DEL CUBO
1110 DATA  50, 50, 50: REM ----------/1
1120 DATA -50, 50, 50: REM I 4 PUNTI  /2
1130 DATA -50,-50, 50: REM SUPERIORI  /3
1140 DATA  50,-50, 50: REM ----------/4
1150 DATA  50, 50,-50: REM ----------/5
1160 DATA -50, 50,-50: REM I 4 PUNTI  /6
1170 DATA -50,-50,-50: REM INFERIORI  /7
1180 DATA  50,-50,-50: REM ----------/8
```

La sezione successiva del programma carica i dati appena definiti nelle matrici X( ), Y( ) e Z( ). Si noti che, per cubi piccoli, i valori iniziali vengono ridotti di un fattore 0,3

```
1190 REM** CARICA I DATI PER 3 CUBI **
1200 FOR I=1 TO 8: REM CUBO GRANDE CENTRALE
1210 READ X(I),Y(I),Z(I)
1220 NEXT
1230 RESTORE
1240 FOR I=9 TO 16: REM CUBO PICCOLO DI DESTRA
1250 READ X,Y,Z
1260 X(I)=.3*X+120: Y(I)=.3*Y: Z(I)=.3*Z
1270 NEXT
1280 RESTORE
1290 FOR I=7 TO 24: REM CUBO PICCOLO DI SINISTRA
1300 READ X,Y,Z
1310 X(I)=.3*X-120: Y(I)=.3*Y: Z(I)=.3*Z
1320 NEXT
```

La sezione seguente ruota di 45° verso destra le coordinate dei punti per i tre cubi

```
1330 REM** RUOTA LE FIGURE DI PI GRECO/4 SULL'ASSE X **
1340 FOR I=1 TO NP
1350 Y(I)=Y(I)*COS(   /4)-Z(I)*SIN(   /4)
1360 Z(I)=Z(I)*COS(   /4)+Y(I)*SIN(   /4)
1370 NEXT
1380 REM** RUOTA LE FIGURE DI PI GRECO/4 SULL'ASSE Z **
1390 FOR I=1 TO NP
1400 X(I)=X(I)*COS(   /4)-Y(I)*SIN(   /4)
1410 Y(I)=Y(I)*COS(   /4)+X(I)*SIN(   /4)
1420 NEXT
```

La prossima sezione del programma carica nella matrice delle connessioni E%(,) i punti che devono essere collegati all'interno della figura per ciascuno dei tre cubi

```
1430 REM** COLLEGAMENTO DEGLI SPIGOLI **
1440 REM -- CUBO GRANDE CENTRALE --
1450 E%(1,2)=1: REM 1 COLLEGATO A 2
1460 E%(2,3)=1: E%(3,4)=1: E%(4,1)=1
1470 E%(5,6)=1: REM QUADRATO INFERIORE
1480 E%(6,7)=1: E%(7,8)=1: E%(8,5)=1
1490 E%(5,1)=1: REM SPIGOLI LATERALI
1500 E%(6,2)=1: E%(7,3)=1: E%(8,4)=1
1510 REM----------------------
1520 REM ** CUBO PICCOLO DI DESTRA **
1530 FOR I=9 TO 16: FOR J=9 TO 16
1540 E%(I,J)=E%(I-8,J-8)
1550 NEXT: NEXT
1560 REM ** CUBO PICCOLO DI SINISTRA **
1570 FOR I=17 TO 24: FOR J=17 TO 24
1580 E%(I,J)=E%(I-8,J-8)
1590 NEXT: NEXT
1600 REM ** PONE IN SIMMETRIA E%(I,J) **
1610 FOR I=1 TO NP: FOR J=1 TO NP
1620 IF E%(I,J)<>0 THEN E%(J,I)=1
1630 NEXT: NEXT
```

Infine il programma chiama il codice macchina che esegue i calcoli di rotazione necessari e traccia la nuova immagine. Questa chiamata viene ripetuta più volte da un ciclo che ruota i tre cubi di 360° prima di terminare

```
1640 REM*******************************
1650 REM** TRACCIA IL CUBO RUOTATO **
1660 SA=2*    /45: CS=COS(SA): SN=SIN(SA)
1670 GOSUB 1790: REM INIZIALIZZA
1680 FOR A=0 TO 2*     STEP SA
1690 SYS 50536: REM ESEGUE LA ROTAZIONE
1700 NEXT A: REM ANGOLO SUCCESSIVO
1710 GET A$: IF A$="" THEN 1710
1720 REM*******************************
1730 GOSUB 1750: REM CANCELLA LO SCHERMO
1740 END
1750 REM** CANCELLA LO SCHERMO **
1760 POKE 49408,0: SYS 49422
1770 PRINT CHR$(147)
1780 RETURN
1790 REM** INIZIALIZZA ROTSUB **
1800 REM*****************************************
1810 REM** N.B. NON DEFINIRE NUOVE VARIABILI **
1820 REM** TRA QUESTA SUBROUTINE E LA        **
1830 REM**   CHIAMATA DELLA ROTAZIONE IN     **
1840 REM**      CODICE MACCHINA. CIO'        **
1850 REM**  POTREBBE CAMBIARE GLI INDIRIZZI  **
1860 REM**       DI BASE DELLE MATRICI       **
1870 REM*****************************************
1880 X(1)=X(1): REM X(1) VAR. CORRENTE
1890 POKE 50500,PEEK(71): REM X(1)LO
1900 POKE 50501,PEEK(72): REM X(1)HI
1910 Y(1)=Y(1): REM Y(1) VAR. CORRENTE
1920 POKE 50502,PEEK(71): REM Y(1)LO
1930 POKE 50503,PEEK(72): REM Y(1)HI
1940 Z(1)=Z(1): REM Z(1) VAR. CORRENTE
1950 POKE 50504,PEEK(71)
1960 POKE 50505,PEEK(72)
1970 POKE 50506,NP: REM NUMERO DI PUNTI
1980 CS=CS: REM CS = VAR. CORRENTE
1990 POKE 50507,PEEK(71)
2000 POKE 50508,PEEK(72)
2010 SN=SN: REM SN = VAR. CORRENTE
2020 POKE 50509,PEEK(71)
2030 POKE 50510,PEEK(72)
2040 E%(1,1)=E%(1,1): REM E% = VAR. CORRENTE
2050 POKE 50511,PEEK(71)
2060 POKE 50512,PEEK(72)
2070 RETURN
```

DATABASE

L'ottava tabella della mappa di memoria del Commodore 64. Per concessione della Commodore Business Machines

INDIRIZZO ESADECIMALE	ETICHETTA	LOCAZIONE DECIMALE	DESCRIZIONE
0283-0284	MAMSIZ	643-644	**Pointer: Top of Memory for O.S.** Puntatore: indirizzo di memoria più alto per il sistema operativo
0285	TIMOUT	645	**Flag: Kernal Variable for IEEE Timeout** Flag: Variabile del Kernal per segnalare "tempo scaduto" sulla linea IEEE
0286	COLOR	646	**Current Character Color Code** Codice di colore del carattere corrente
0287	GDCOL	647	**Background Color Under Cursor** Colore di sfondo sotto il cursore
0288	HIBASE	648	**Top of Screen Memory (Page)** Cima della memoria video (valore della pagina)
0289	XMAX	649	**Size of Keyboard Buffer** Dimensioni del buffer per la tastiera
028A	RPTFLG	650	**Flag: REPEAT Key Used, $80 = Repeat** Flag: Ripetizione Tasto attiva
038B	KOUNT	651	**Repeat Speed Counter** Contatore di velocità per la ripetizione dei tasti
038C	DELAY	652	**Repeat Delay Counter** Contatore per il ritardo nella ripetizione dei tasti
028D	SHFLAG	653	**Flag: Keyboard SHIFT Key/CTRL Key/C = Key** Flag: tasti SHIFT, CTRL e C = attivi
028E	LSTSHF	654	**Last Keyboard Shift Pattern** Ultima configurazione di shift per la tastiera
028F-0290	KEYLOG	655-656	**Vector: Keyboard Shift Pattern** Vettore: Configurazione di shift per la tastiera
0291	MODE	657	**Flag: $00 = Disable SHIFT Keys, $80 = Enable** Flag di abilitazione / diabilitazione dei tasti SHIFT; $00 = Disabilitati, $80 = attivi
0292	AUTODN	658	**Flag: Auto Scroll Down, 0 = On** Flag: Scroll automatico verso il basso, 0 = attivo
0293	M51CTR	659	**RS-232: 6551 Control Register Image** RS-232: Immagine del registro di controllo del 6551
0294	M51CDR	660	**RS-232: 6551 Command Register Image** RS-232: Immagine del registro di comando del 6551
0295-0296	M51AJB	661-662	**RS-232: Non Standard BPS (Time/2-100) USA** RS-232: Velocità non standard
0297	RSSTAT	663	**RS-232: 6551 Status Register Image** RS-232: Immagine del registro di stato del 6551
0298	BITNUM	664	**RS-232: Number of Bits Left to Send** RS-232: Numero di bit ancora da trasmettere

Metodi Scout e B-Star

Strategie di gioco alternative per l'Intelligenza Artificiale

La procedura alfa-beta, esaminata in precedenza, senz'altro più efficace di quella mini-max poiché identifica e rimuove tutti i rami superflui dall'albero di gioco, è stata adottata con successo in moltissimi programmi per il gioco degli scacchi. Recentemente, però, sono stati ideati altri due metodi alternativi: uno è l'algoritmo "Scout" (formulato da Judea Pearl) e l'altro è l'algoritmo "B-Star" (o B*) di Hans Berliner.

Il principio del metodo Scout è quello di eliminare, senza ulteriori esami, le mosse inaccettabili adoperando una funzione di valutazione altamente specializzata. Soltanto le mosse più promettenti, dunque, vengono esaminate a fondo.

Nel metodo B*, invece, viene prima effettuato un esame dei livelli superiori dell'albero tentando, nel più breve tempo possibile, di raggiungere uno dei due seguenti scopi:

- Provare che la migliore mossa apparente è effettivamente la migliore mossa reale.
- Provare che nessuna delle mosse alternative è migliore.

Questa strategia bivalente si ottiene implementando due procedure chiamate ProveBest (per dimostrare la migliore mossa) e RefuteRest (per eliminare le restanti), che assegnano a ciascun nodo due valori: uno per la stima "ottimistica" e l'altro per quella "pessimistica". Lo scopo è quello di costringere l'algoritmo di ricerca ad occuparsi della zona dell'albero dove si riscontra maggiore incertezza e dove tale incertezza può influire pesantemente sulla decisione finale.

Scomposizione della ricerca

Sarebbe errato concludere che la ricerca basata sulla struttura ad albero sia l'unica adottabile nei giochi al computer: esistono giochi in cui la filosofia della ricerca risulta ridotta in rami elementari, nonostante la presenza di ingegnosi trucchi per velocizzarne lo svolgimento. Tra questi giochi si possono citare i più comuni giochi di carte (in particolare Bridge e Poker) e numerosi giochi da scacchiera (quali il Go e il Go-moku).

Questi giochi sono caratterizzati da vari livelli di difficoltà e vincere richiede ai giocatori "umani" una buona dose di intelligenza. I tentativi di adottare stategie "ad albero", per



Una futile ricerca

Sebbene alcuni giochi possano essere "intelligentemente" giocati dal computer, avvalendosi di sofisticate strategie di ricerca, in altri i vari metodi basati sulla ricerca producono risultati davvero scarsi. Ciò può essere dovuto alla presenza di fattori casuali (come nel "Backgammon") oppure a una troppo ampia diramazione dell'albero di gioco, che il computer non è in grado di gestire in una sola volta. Occorrono quindi algoritmi alternativi per risolvere certi tipi di giochi

IL GERGO DELLA RICERCA AD ALBERO

Albero di gioco	La struttura ad albero, formata considerando tutte le possibili mosse seguite da quelle dell'avversario, e così via
Previsione	Il processo di costruzione dell'albero
Valore "ritroso"	Il valore assegnato a un nodo di un albero di gioco, esaminando i valori sottostanti, partendo dal basso verso l'alto
Minimassimizzazione	Procedimento di scelta del valore ritroso, in base alla minimizzazione delle pile dispari (mosse avversarie) e alla massimizzazione delle pile pari (mosse del giocatore)
Algoritmo Alfa-Beta	Un miglioramento del metodo "minimax", che elimina le porzioni dell'albero di gioco che non hanno alcuna influenza sul risultato al livello superiore
Fattore di diramazione	Il numero medio di diramazioni o mosse a ciascun livello dell'albero di gioco. Il gioco orientale "Go" possiede un fattore di diramazione superiore a 200

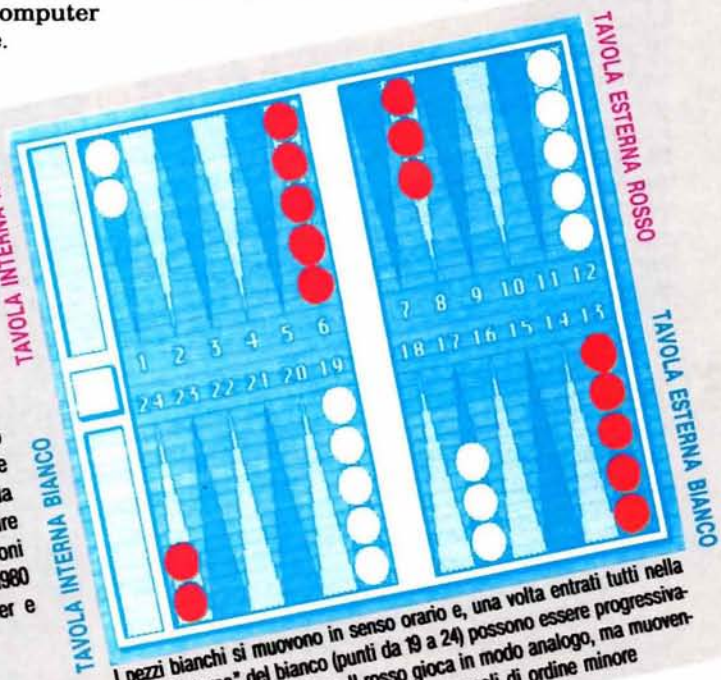
il gioco con computer, sono però falliti. Una delle ragioni di ciò è molto semplice: il fattore di diramazione è talmente alto (ossia le mosse possibili sono così tante), che il computer non riesce a valutarle tutte insieme.

Un altro motivo di insuccesso è che gli algoritmi per la ricerca ad albero sono soltanto un'approssimazione del processo umano usato per risolvere problemi: gli esperti consultano, mentalmente, il proprio albero di gioco in modo spesso irrazionale e talvolta tenendo conto di fattori tipicamente umani.
Hans Berliner, l'ideatore del metodo B*, ha sviluppato un programma per Backgammon che, nel 1980, ha sconfitto il campione mondiale di questo gioco. Nonostante ciò, il programma non esegue alcuna ricerca, almeno

gioco. Non esistono elementi casuali, ma il fattore di diramazione è talmente grande che non nel senso convenzionale. Alla base del Backgammon c'è un elemento probabilistico (il lancio dei dadi), che introduce ramificazioni direttamente controllate da ciascun giocatore a turno. Ciò rende quasi impossibile l'applicazione delle strategie di previsione fin qui considerate.
L'efficienza del programma di Berliner consiste nell'alta capacità (certamente superiore a quella umana) di calcolare le possibilità offerte dal lancio dei dadi e dalle combinazioni delle pedine che possono verificarsi durante il gioco. Inoltre, il programma possiede una funzione di valutazione molto sofisticata, composta in effetti da un gruppo di funzioni "specializzate" per ogni fase del gioco.
Un altro gioco, spesso usato per collaudare programmi di AI, proviene dall'Oriente. Si tratta del Go, che si gioca muovendo sassolini (o pedine) su di una scacchiera composta da 18 x 18 caselle. Scopo del gioco è quello di occupare sezioni della scacchiera e di accerchiare le pedine avversarie per eliminarle dal

DENTRO LA TAVOLA INTERNA

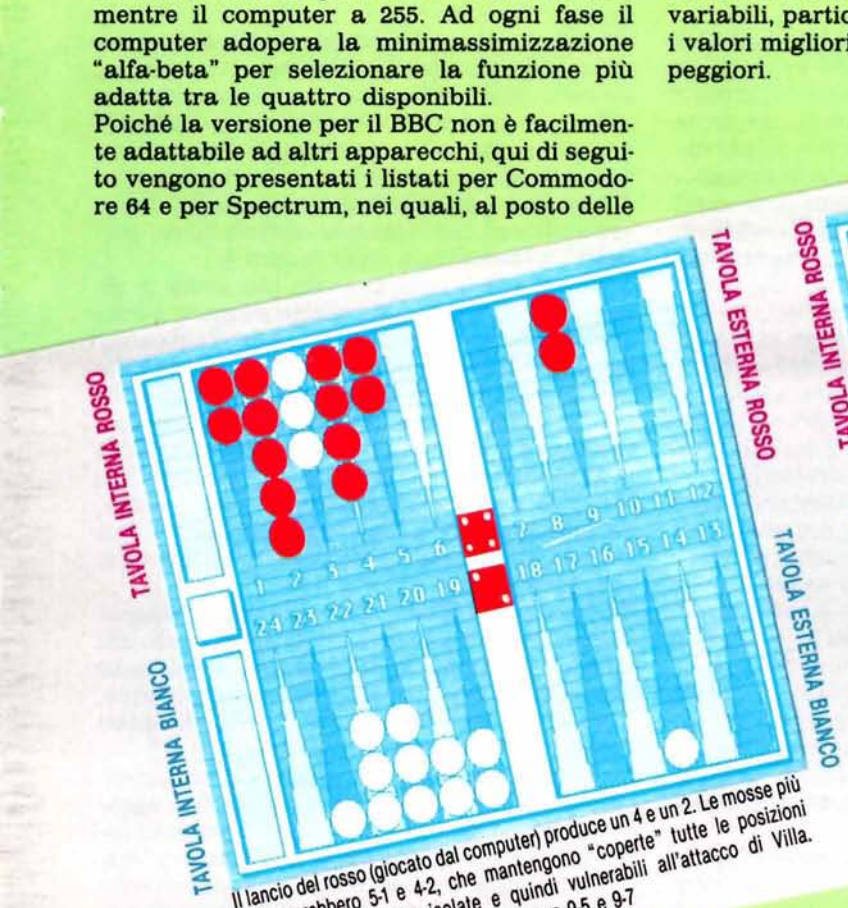
Il Backgammon si gioca su un tabellone sul quale sono riprodotti 24 triangoli. Su questi vengono allineate le pedine, in base al risultato di un lancio di dadi. Scopo del gioco è quello di far raggiungere alle proprie pedine il lato opposto, facendole così uscire "indenni" dal gioco, prima che ci riesca l'avversario. Se i valori di ciascun dado sono diversi, si può muovere un solo pezzo di un numero di posizioni pari alla somma dei dadi oppure spostare due pedine di tanti posti quanti sono, rispettivamente, i punti di ciascun dado. Nel primo caso, si considera il movimento come composto da due mosse distinte. Se in una particolare posizione esiste una sola pedina e la posizione viene occupata da una pedina avversaria, la prima esce dal gioco e va fatta ripartire dall'inizio, prima di passare alla mossa successiva. Le illustrazioni mostrano alcune mosse della partita giocata a Montecarlo nel 1980 tra il programma per Backgammon ideato da Hans Berliner e l'allora campione del mondo Luigi Villa



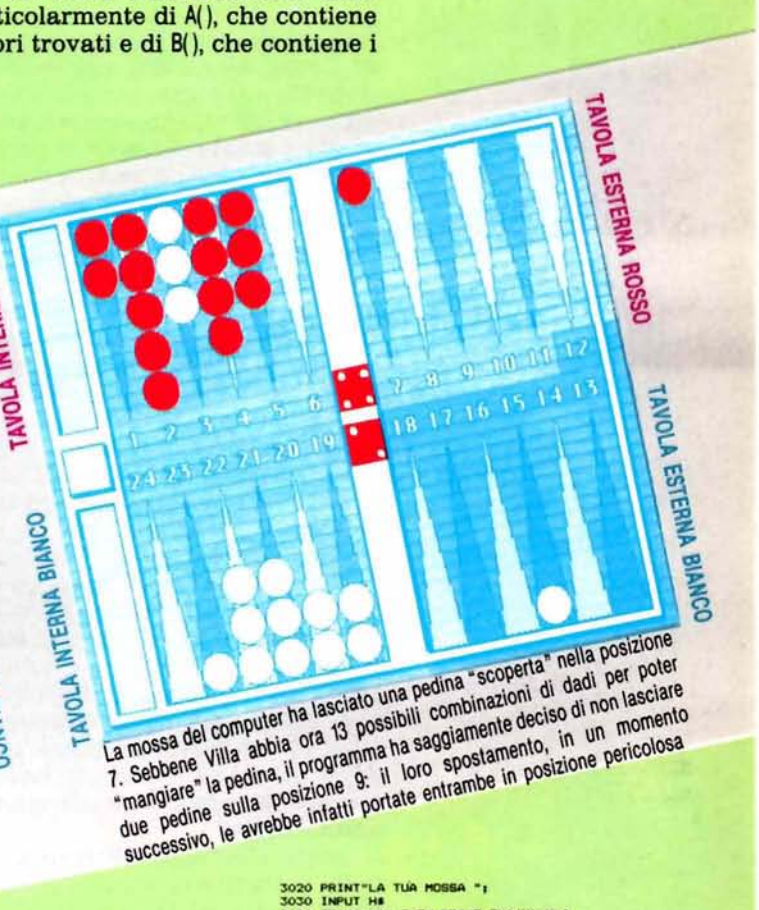
I pezzi bianchi si muovono in senso orario e, una volta entrati tutti nella tavola "interna" del bianco (punti da 19 a 24) possono essere progressivamente rimossi dal tabellone. Il rosso gioca in modo analogo, ma muovendosi in senso antiorario, ossia verso i triangoli di ordine minore

le tecniche basate sulla ricerca risultano totalmente inutili.
I migliori programmi per Go considerano la scacchiera in termini di unità il cui valore è superiore a quello delle singole pedine. Il gioco del Go con computer è molto arretrato rispetto a quello degli scacchi e ciò forse dipende da un'inadeguata definizione di cosa sia la percezione umana. Forse i Giapponesi, dalla cui mentalità è scaturito il gioco, riusciranno meglio nel tentativo, magari con i computer della Quinta Generazione.
Esistono, è vero, programmi per giocare a Go sui comuni home computer, ma sono ben lontani dal poter battere i maestri del gioco.

Per illustrare i principali concetti della ricerca ad albero, è stato presentato (pag. 1751) un programma BASIC per il Micro BBC fondato, in pratica, sulla ricerca pura e sull'ampio uso di funzioni ricursive e di parametri.
Il programma consente (al giocatore e al computer, a turno) di scegliere tra quattro diverse funzioni che modificano un valore esistente per produrne uno nuovo. Il giocatore tenta cioè di portare il valore a -255, mentre il computer a 255. Ad ogni fase il computer adopera la minimassimizzazione "alfa-beta" per selezionare la funzione più adatta tra le quattro disponibili.
Poiché la versione per il BBC non è facilmente adattabile ad altri apparecchi, qui di seguito vengono presentati i listati per Commodore 64 e per Spectrum, nei quali, al posto delle funzioni ricursive, compaiono chiamate del tipo GOSUB. I parametri e le variabili locali (eccetto D, il contatore di profondità), sono stati sostituiti da matrici, dimensionate alla linea 1100. La variabile D funziona come una specie di puntatore di stack, poiché tiene il conto di quale elemento delle matrici è adoperato durante ciascuna fase del gioco.
Rispetto alla versione BBC, la principale differenza è la vasta indicizzazione delle variabili, particolarmente di A(), che contiene i valori migliori trovati e di B(), che contiene i peggiori.



Il lancio del rosso (giocato dal computer) produce un 4 e un 2. Le mosse più ovvie sarebbero 5-1 e 4-2, che mantengono "coperte" tutte le posizioni senza lasciare pedine isolate e quindi vulnerabili all'attacco di Villa. Tuttavia, il computer decise di muovere 9-5 e 9-7



La mossa del computer ha lasciato una pedina "scoperta" nella posizione 7. Sebbene Villa abbia ora 13 possibili combinazioni di dadi per poter "mangiare" la pedina, il programma ha saggiamente deciso di non lasciare due pedine sulla posizione 9: il loro spostamento, in un momento successivo, le avrebbe infatti portate entrambe in posizione pericolosa

IL GIOCO DEI NUMERI

```
80 GOSUB 1000: REM INIZIALIZZAZIONE
90 GOSUB 1600: REM ISTRUZIONI
100 :
110 REM *** CICLO PRINCIPALE ***
120 GOSUB 2000: REM PREPARA UN NUOVO GIOCO
130 INPUT "CHI PARTE PER PRIMO (1=TU, 2=IO)"; H1
140 IF H1<1 OR H1>2 THEN 130
150 REM *** CICLO DEL GIOCO ***
160 IF H1=1 THEN GOSUB 3000
170 REM *** TURNO DEL GIOCATORE ***
180 GOSUB 3500: REM VISUALIZZA LA TABELLA
190 H1=1: REM SEMPRE 1 DOPO IL 1MO CICLO
200 GOSUB 4000: REM VERIFICA DELLA VITTORIA
210 IF EG=0 THEN GOSUB 5000
220 REM *** TURNO DEL COMPUTER ***
230 GOSUB 3500: REM VISUALIZZA LO STATO DEL GIOCO
240 GOSUB 4000: REM VERIFICA DI FINE GIOCO
250 IF EG=0 AND M<=33 THEN 150: REM CHIUDE IL CICLO
260 REM *** FINALE ***
270 GOSUB 6000: REM CONGRATULAZIONI
280 INPUT"GIOCHI ANCORA (1=SI,2=NO)";S
290 IF S<1 OR S>2 THEN 280
300 IF S=1 THEN 110: REM NUOVO GIOCO
310 PRINT: PRINT"CIAO E GRAZIE PER LA PARTITA!"
320 END
330 :
500 REM *** MASSIMIZZA ***
510 D=D+1: C1=C1+1
520 IF D>=MD OR ABS(V(D))>HI THEN A(D)=V(D): D=D-1: RETURN
530 REM *** ALTRIMENTI VA PIU' A FONDO ***
540 P(D)=0
550 REM *** ATTRAVERSO L'ALBERO ***
560 P(D)=P(D)+1: H=P(D): V=V(D): GOSUB 5500: REM ESEGUE MOSSA
570 IF D=1 THEN PRINTCHR$(64+H);" = " ;
580 D1=D+1
590 A(D1)=A(D):B(D):V(D1)=V:GOSUB 700:REM CHIAMA MINIMIZZAZIONE
600 IF B(D+1)>A(D) THEN A(D)=B(D+1):K(D)=P(D)
610 IF D=1 THEN PRINTB(D+1);": ";
620 IF P(D)<=3 AND A(D)<B(D) THEN 550
630 IF D=1 THEN BV=A(D): HH=K(D): REM CONSERVA IL MIGLIORE
640 D=D-1: RETURN
650 :
700 REM *** MINIMIZZAZIONE ***
710 D=D+1: C2=C2+1
720 IF D>=MD OR ABS(V(D))>HI THEN B(D)=V(D): D=D-1: RETURN
730 P(D)=0
740 REM *** ATTRAVERSO L'ALBERO ***
750 P(D)=P(D)+1: H=P(D):V=V(D): GOSUB 5500: REM ESEGUE MOSSA
760 D1=D+1:A(D1)=A(D):B(D1)=B(D):V(D1)=V
770 GOSUB 500: REM CHIAMA MASSIMIZZAZIONE
780 IF A(D+1)<B(D) THEN B(D)=A(D+1)
790 IF P(D)<=3 AND B(D)>A(D) THEN 740
800 D=D-1: RETURN
810 :
1000 REM *** INIZIALIZZAZIONE ***
1010 BL$=""
1020 REM *** DEFINIZIONE DELLE QUATTRO FUNZIONI **
1030 DEF FNA(X)=2*X-7
1040 DEF FNB(X)=INT(X/2)+1
1050 DEF FNC(X)=4*X+17
1060 DEF FND(X)=3*X-4
1070 LO=-255: HI=255
1080 REM *** MATRICI USATE DA MINIMAX ***
1090 D=16
1100 DIM V(D), A(D), B(D), P(D), K(D)
1110 RETURN
1120 :
1600 REM *** ISTRUZIONI ***
1610 PRINT"BENVENUTO AL GIOCO DEI NUMERI"
1620 PRINT"IO PROVO A MASSIMIZZARE, IL TUO"
1630 PRINT"COMPITO E' MINIMIZZARE"
1640 PRINT"PER VEDERE L'EFFETTO DI UNA MOSSA DIGITA:"
1650 PRINT"A, B, C OPPURE  D. IL TASTO X LA ESEGUE"
1660 PRINT: RETURN
1670 :
2000 REM *** PREPARAZIONE ***
2010 M=0: V=INT(RND(1)*15)-8: REM STATO INIZIALE
2020 EG=0
2030 PRINT"STATO INIZIALE = ";V
2040 RETURN
2050 :
3000 REM *** MOSSA DEL GIOCATORE ***
3010 M=M+1 :PRINT
3020 PRINT"LA TUA MOSSA ";
3030 INPUT H$
3040 IF H$="A" THEN PRINT FNA(V):H=1
3050 IF H$="B" THEN PRINT FNB(V):H=2
3060 IF H$="C" THEN PRINT FNC(V):H=3
3070 IF H$="D" THEN PRINT FND(V):H=4
3080 IF H$<>"X" THEN 3020:REM SCELTA NON VALIDA
3090 GOSUB 5500: REM ESEGUE MOSSA
3100 RETURN
3110 :
3500 REM *** VISUALIZZA LA TABELLA ***
3510 PRINT: PRINT"    MOSSA";M;" -->";
3520 IF M<1 THEN RETURN
3530 PRINT CHR$(64+H);
3540 PRINT" = ";V:PRINT:RETURN
3550 :
4000 REM *** VERIFICA DELLA VITTORIA ***
4010 IF M<1 THEN RETURN
4020 EG=0
4030 IF V<LO THEN EG=-1
4040 IF V>HI THEN EG=1
4050 RETURN
4060 :
5000 REM *** MOSSA DEL COMPUTER ***
5010 W=V: REM CONSERVA LA STATO ATTUALE
5020 M=M+1
5030 MD=6:REM MASSIMA PROFONDITA'
5040 IF M<4 THEN MD=4
5050 IF M>8 THEN MD=8
5060 GOSUB 5200: REM ---> H
5070 V=W: REM RIPRISTINA LA STATO
5080 GOSUB 5500: REM ESEGUE MOSSA
5090 RETURN
5100 :
5200 REM *** SELEZIONE DELLA MOSSA ***
5210 BV=LO:D=0
5220 V(1)=V:A(1)=LO:B(1)=HI
5230 GOSUB 500: REM MASSIMIZZA
5240 H=HH
5250 PRINT: INPUT"PREMI RETURN PER CONTINUARE"; Q
5260 RETURN
5270 :
5500 REM *** ESEGUE MOSSA ***
5510 IF H=1 THEN V=FNA(V): RETURN
5520 IF H=2 THEN V=FNB(V): RETURN
5530 IF H=3 THEN V=FNC(V): RETURN
5540 IF H=4 THEN V=FND(V): RETURN
5550 :
6000 REM *** CONGRATULAZIONI ***
6010 PRINT:PRINT"GAME OVER"
6020 IF EG>0 THEN PRINT"HO VINTO IO!"
6030 IF EG<0 THEN PRINT"HAI VINTO TU!"
6040 IF EG=0 THEN PRINT"PAREGGIO!"
6050 RETURN
```

I comandi "transienti"

Una descrizione dei comandi messi a disposizione dal CP/M della Digital Research

Nella prima parte dedicata al sistema operativo CP/M, ne è stata esaminata la struttura generale e si è visto come le routine per parte dei comandi vengano conservate permanentemente in RAM mentre altre, normalmente memorizzate su disco, sono caricate in memoria soltanto quando occorrono per poi venir eliminate. Queste routine, che formano i cosiddetti comandi "transienti" del CP/M, consentono di comunicare con le unità a dischetti, di gestire i file e controllare le varie periferiche.

All'accensione, il computer esegue una serie di verifiche interne, per assicurarsi che tutte le componenti necessarie siano adeguatamente collegate e apre, se necessario, anche le porte di comunicazione con le periferiche connesse (inviando o ricevendo speciali caratteri di controllo). Tutta questa fase preliminare viene chiamata "inizializzazione" del sistema.

Nell'inizializzare alcune periferiche, può darsi che il computer si fermi ad attendere un messaggio di conferma da parte di esse, prima di procedere oltre. Un esempio si ha quando si innestano cartucce di programmi, che spesso prendono il controllo dell'apparecchio in tale fase, provvedendo direttamente a tutte le funzioni di input/output, senza passare per la ROM del BASIC. In modo analogo, il CP/M attende un segnale di conferma da parte delle unità a dischi.

Il primo risultato dell'inizializzazione delle unità a dischi è il posizionamento della testina di lettura/scrittura e l'avvio della lettura della prima traccia (detta "di sistema") del dischetto. In genere, se il dischetto non risulta inserito, il sistema si ferma in stato di attesa, oppure invita l'utente a provvedere, tramite un opportuno messaggio sullo schermo. La traccia 0 dei dischetti per CP/M contiene il cosiddetto "bootstrap loader", ossia il programma caricatore di base, che provvede a sua volta al caricamento in memoria del resto del CP/M. Se il caricatore non viene trovato sul dischetto, l'errore viene segnalato.

Estensione	Commento	Esempio
ASM	File sorgente in Assembly	CODPROG.ASM
BAK	File di archivio creato da un Programma editor	MEMO.BAK
BAS	Programma in BASIC	PROG.BAS
COM	Programma eseguibile	PIP.COM
HEX	File contenente codice esadecimale	GRAFICA.HEX
INT	File di codice intermedio (usati dai compilatori)	GIOCO.INT
PRN	File documentativo prodotto da un compilatore o un assembler	CODPROG.PRN
SUB	File per esecuzione di comandi in sequenza (batch)	SPOOLER.SUB
$$$	File di appoggio temporaneo, creato dal programma editor	TESTO.$$$

Terminato il completo caricamento del CP/M, sullo schermo appare il simbolo A> (oppure, su certi apparecchi, 0A>), detto "prompt", seguito dal cursore. Questo indica che il sistema è in attesa di un comando e informa, inoltre, quale sia il drive attualmente in funzione (su cui vengono effettuate le operazioni di lettura/scrittura). Nell'esempio citato il drive "corrente" è identificato dalla lettera A.

In un sistema composto da più unità a dischetti, può essere necessario passare ad un altro drive, ad esempio il B. Ciò si ottiene semplicemente digitando la lettera B, seguita da un "due punti" e dalla pressione del tasto Return. Il CP/M tenta un accesso al disco e comunica l'avvenuto scambio visualizzando il simbolo B> sullo schermo. Il CP/M consente l'accesso fino a 16 drive (purché, ovviamente, collegati), identificati dalle lettere da A a P, ma in genere il numero massimo di drive è limitato a 4 (da A a D).

Se le unità a dischi sono del tipo a doppia faccia, il sistema può essere programmato dal fabbricante in modo da accedere a ciascuna faccia come se si trattasse di un singolo drive, cosicché le due facciate di un singolo disco vengono chiamate, rispettivamente, A e B.

La prima operazione successiva al caricamento, di solito, è l'accertare quali file esistano sul dischetto "corrente", esaminando la sezione del disco che contiene la "directory" (ossia il catalogo dei file). Il comando occorrente è DIR (oppure dir, dato che il CP/M non distingue tra caratteri minuscoli e maiuscoli). L'effetto risultante è la visualizzazione dell'elenco dei file, compresi quelli caricabili ed eseguibili dal CP/M (identificati dall'estensione .COM).

Dover caricare un comando in memoria, per poi eseguirlo, può sembrare un procedimento piuttosto tortuoso ma, oltre che per risparmiare memoria, serve per aumentare la compatibilità tra apparecchi di marca diversa.

Come è facile intuire, il CP/M contiene un numero di comandi per l'accesso ai file conservati su disco e, sebbene i programmi applicativi debbano possedere un formato standard per essere utilizzabili anche su computer diversi, i fabbricanti di hardware hanno spesso "personalizzato" alcuni dei comandi per adattarli ai propri apparecchi. Un caso tipico è il comando di formattamento dei dischetti, per il quale non è mai stato fissato uno standard, cosicché ogni fabbricante fornisce, sul dischetto di sistema, un apposito comando transiente.

Come accennato in precedenza, ogni nome di file consiste in due parti distinte: il nome vero

e proprio (lunghezza massima 8 caratteri) e, separata da un punto, la cosiddetta "estensione" (composta da un massimo di tre caratteri). L'estensione influisce sul modo usato per caricare in memoria il file. Sul dischetto di sistema esistono svariati file che contengono comandi transienti, la cui estensione è .COM. Tali file sono eseguibili semplicemente digitandone il nome (senza specificare l'estensione) e premendo Return.

Il comando STAT

Il comando interno DIR, visto poc'anzi, è strettamente correlato ad un comando transiente noto col nome STAT, che fornisce ulteriori informazioni sul dischetto e sui file in esso contenuti. Digitando semplicemente STAT, viene visualizzata la quantità di memoria attualmente libera sul dischetto e la corrente condizione di lettura/scrittura. R/W, ad esempio, indica un dischetto sul quale è abilitata sia la lettura che la scrittura, mentre R/O indica che sul dischetto è consentita la sola lettura (il disco è protetto contro la scrittura).
Per ottenere questa seconda condizione, si usa lo stesso comando STAT ma nel formato: STAT D: $R/O, dove D rappresenta la lettera identificativa del drive. Qualsiasi successivo tentativo di scrivere sul disco provoca la segnalazione di un errore di tipo BDOS. Il comando STAT può anche servire per esaminare o modificare la condizione di lettura/scrittura di un singolo file: basta indicare, in aggiunta al nome del drive quello del file. Digitando STAT seguito dal solo nome del file (estensione compresa) vengono visualizzati dati sull'occupazione in Kbyte del file specificato. Inoltre, il comando STAT può essere usato per esaminare o modificare lo stato delle periferiche. Per esempio, il comando STAT DEV visualizza un elenco di tutti i dispositivi di input/output connessi (compreso lo schermo).
Una delle più importanti caratteristiche di un sistema operativo è la possibilità di trasferire file da un drive all'altro. In CP/M questa operazione si esegue tramite il comando transiente PIP (Peripheral Interchange Program, o programma di interscambio tra periferiche). La funzione del comando va oltre la semplice copia di un file da un disco all'altro: consente anche l'invio di file alla stampante, allo schermo o ad altri dispositivi.
Il modo più comodo per usare il comando PIP è quello di digitarne il nome e premere Return. Sullo schermo appare un nuovo prompt, un semplice asterisco, a significare che il programma è in attesa di ulteriori specifiche. Il formato di queste è: D:NOMEDEST = D:NOMESORG. Supponiamo di aver già richiamato il PIP e di avere un file chiamato GIANNI.TXT, da copiare su di un altro disco assegnandogli il nome MARCO.TXT. Per prima cosa occorre inserire nel drive A il disco contenente il file GIANNI.TXT e nel drive B il dischetto sul quale eseguire la copia. A questo punto si impartisce il comando B:MARCO.TXT = A:GIANNI.TXT.
Si noti che, nell'impiego di PIP, il nome del file di destinazione viene specificato per primo e che i nomi vanno digitati per intero (estensione inclusa). Quando sul video appare un nuovo asterisco l'operazione è conclusa e basta premere Return per tornare al prompt del CP/M. Per verificare la presenza di MARCO.TXT sul dischetto B si può usare il comando DIR B:.
In questo esempio, oltre a trasferire il file, ne è stato cambiato anche il nome, anche se ciò non è affatto obbligatorio. Anzi, se il nome resta identico il comando può essere abbreviato in B: = A:GIANNI.TXT.
Il PIP è utilizzabile anche se si possiede un solo drive: in tal caso il programma informa quando è il momento di scambiare il dischetto sorgente con quello di destinazione e viceversa.
Volendo inviare GIANNI.TXT alla stampante, si può usare una sequenza di comandi simile: PIP LPT: = B:GIANNI.TXT. Si noti che in questo caso sia il comando PIP sia le specifiche fanno parte di un'unica riga di comando. Il comando PIP (assieme ad alcuni altri) accetta anche tale formato più sbrigativo.
Nel caso specifico, la destinazione non è propriamente un file, bensì un canale di uscita (la stampante, identificata da LPT:). Inoltre, in questo comando si è anche specificato il drive "sorgente" (B:), dal quale estrarre il file. Se il drive sorgente è quello in uso, tale specifica può essere omessa.
Supponiamo adesso di voler soltanto cambiare il nome ad un file, senza trasferirlo. Invece di usare il PIP, è più comodo adoperare l'apposito comando interno REN (da "REName"). Se vogliamo cambiare GIANNI.TXT in MARCO.TXT basta impartire il comando REN MARCO.TXT = GIANNI.TXT, rammentando che il nome del file destinazione, anche in questo caso, precede sempre quello del file sorgente. Un comando DIR può rassicurarci dell'avvenuta modifica del nome.
Infine, può esser necessario cancellare un file da un dischetto. Per far ciò si ricorre al comando ERA (da "ERAse", cancellare), seguito dal nome completo del file, ad esempio: ERA GIANNI.TXT. Questo comando va impartito con grande attenzione, poiché il file, una volta cancellato, non è recuperabile. In particolare, si presti cura nello specificare il drive sul quale effettuare l'operazione, se è diverso da quello "corrente", ad esempio: ERA B:MARCO.TXT.

ASM - L'ASSEMBLATORE DEL CP/M

Nella maggior parte delle implementazioni, il dischetto di sistema del CP/M è corredato da un programma assembler, chiamato ASM.COM. Benché poco noto tra i programmatori dediti al BASIC, questo è un programma utilissimo e piuttosto sofisticato per la codifica di programmi in codice macchina.
L'unico inconveniente del programma ASM è che accetta soltanto i listati sorgenti scritti per il processore 8080, nella maggior parte dei casi oggi sostituito dal più efficiente Z80. Nonostante il codice "oggetto" generato da questo assemblatore funzioni regolarmente anche sullo Z80, mancano ovviamente tutte le istruzioni relative alle nuove funzioni introdotte da questo microprocessore. La Digital Research, creatrice del CP/M, fornisce a richiesta una serie di definizioni "macro", che consentono la programmazione dello Z80. In alternativa, esistono in commercio ottimi assemblatori per Z80, specificamente progettati per l'uso sotto CP/M

Tracce per la memoria

La calibrazione e il software per il tracciatore digitale

Il primo passo nello sviluppo del software di controllo consiste nella calibrazione dell'apparecchio. Il programma *Calibratore*, presentato qui sotto, prevede quattro valori digitali, corrispondenti alle posizioni "critiche" del Braccio tracciatore: posizione 0 del Braccio 1, posizione 0 del Braccio 2, posizione 90° del Braccio 1 e posizione 90° del Braccio 2. Questi valori vanno immessi nelle linee 1200, 1210, 1240 e 1250, come parte della procedura definisci_parametri. Altri due parametri che possono essere diversi, da una realizzazione all'altra, sono le lunghezze delle due sezioni del Braccio: queste vanno specificate nelle linee 1220 e 1260 del programma di calibrazione. Tali lunghezze vanno espresse in millimetri e che per il Braccio 2 la misura comprende la distanza tra l'estremità e l'incrocio del mirino.

Lettura degli angoli

Stabiliti i valori di calibrazione si possono usare per convertire le letture digitali fornite dai potenziometri (tramite il convertitore analogico/digitale del Micro BBC) in valori angolari che vengono tradotti in distanze verticali e orizzontali del mirino rispetto al perno principale del Braccio. Il valore di un angolo può essere calcolato con:

angolo = (ADVAL(n) - poszero)*90/(posnovanta - poszero)

Poiché la seconda parte di questa espressione è costante, conviene calcolarne il valore all'inizio del programma ed usare il risultato nei vari calcoli. Se si definisce:

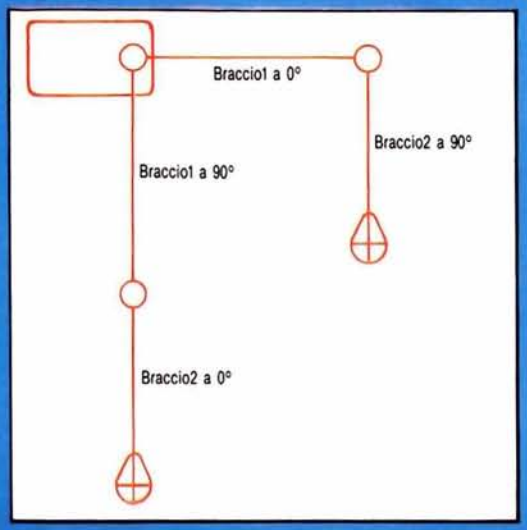
fattore = 90/(posnovanta - poszero)

CALIBRAZIONE

Si disponga il tracciatore in modo che la scatoletta di supporto si trovi in alto a sinistra. Per tarare adeguatamente il dispositivo, occorrono quattro informazioni: la posizione 0° del Braccio 1, la posizione 0° del Braccio 2, la posizione 90° del Braccio 1 e la posizione 90° del Braccio 2. Dopo aver avviato il programma di calibrazione, si muova il dispositivo in modo da portare il Braccio 1 in posizione parallela al bordo superiore della tavoletta e il Braccio 2 in posizione perpendicolare. Si prenda nota dei valori generati dai bracci 1 e 2. Adesso si portino entrambi i bracci in una posizione parallela al bordo laterale della tavoletta e si annotino i valori. Poi si prema un qualsiasi tasto per passare al modo "angoli".

A questo punto si immettano i quattro valori rilevati, nell'ordine indicato dal programma, controllando che i valori visualizzati siano esatti. Si conservino i quattro valori trovati in via sperimentale, poiché servono anche per il programma di controllo per il tracciatore

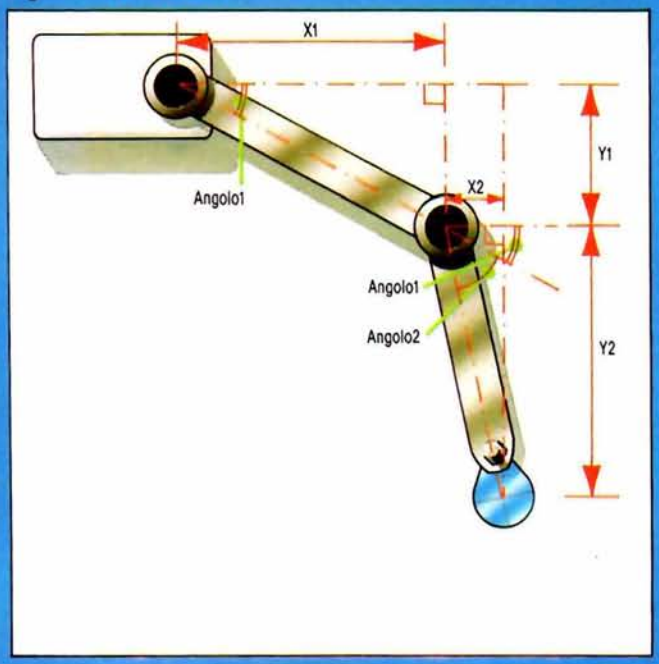
Posizione di calibrazione




```
1000 REM **** TARATURA DEL TRACER - BBC ****
1010 PROCtaratura
1020 PROCvisione_angolo
1030 END
1040 DEF PROCtaratura
1050 CLS
1060 PRINT TAB(5,10);"ANGOLO 1";TAB(20);
     "ANGOLO 2"
1070 REPEAT
1080 PRINT TAB(5,12);SPC(40)
1090 PRINT TAB(5,12);ADVAL(1);TAB(20);ADVAL(2)
1100 PROCattesa(600)
1110 A$=INKEY$(1)
1120 UNTIL A$<>""
1130 ENDPROC
1150 DEF PROCvisione_angolo
1160 PROCdefiniz_angoli
1170 CLS
1180 PRINT TAB(5,10);"ANGOLO 1";TAB(20);
     "ANGOLO 2"
1190 REPEAT
1200 PROCcalcolo_angoli
1210 PRINT TAB(5,12);SPC(40)
1220 PRINT TAB(5,12);angolo1;TAB(20)angolo2
1230 PROCattesa(1000)
1240 A$=INKEY$(1)
1250 UNTIL A$<>""
1260 ENDPROC
1280 DEF PROCdefiniz_angoli
1290 CLS
1300 PRINT TAB(5,10);:INPUT"1mo braccio pos. zero"
     ;uno_zero
1310 PRINT TAB(5);:INPUT"1mo braccio pos. novanta"
     ;uno_novanta
1320 PRINT TAB(5);:INPUT"2do braccio pos. zero"
     ;due_zero
1330 PRINT TAB(5);:INPUT"2do braccio pos. novanta"
     ;due_novanta
1340 fattore_uno=90/(uno_novanta-uno_zero)
1350 fattore_due=90/(due_novanta-due_zero)
1360 ENDPROC
1380 DEF PROCcalcolo_angoli
1390 angolo1=INT((ADVAL(1)-uno_zero)*fattore_uno)
1400 angolo2=INT((ADVAL(2)-due_zero)*fattore_due)
1420 DEF PROCattesa(ritardo)
1430 LOCAL I
1440 FOR I=1 TO ritardo:NEXT I
1450 ENDPROC
1460 PROCattesa(1000)
```

KEVIN JONES

Trigonometria del Braccio



Gli spostamenti orizzontali e verticali del mirino possono essere calcolati in base ai valori degli angoli, forniti dai segnali prodotti dai potenziometri: basta qualche piccola nozione di trigonometria, applicata ai due triangoli rettangoli indicati in figura. Le lunghezze x1 e y1 corrispondono alla lunghezza del Braccio 1 moltiplicata, rispettivamente, per COS(angolo1) e SIN(angolo1). Poiché il valore fornito dal secondo potenziometro è relativo alla posizione del Braccio 1, il calcolo delle lunghezze x2 e y2 è un po' più complesso. L'angolo da considerare, in questo caso, non è l'angolo2, bensì la somma angolo1+angolo2. Pertanto COS e SIN vanno applicati a tale somma. Le distanze totali si ottengono sommando, rispettivamente, x1 con x2 e y1 con y2

KEVIN JONES

la formula può essere riscritta così:

```
angolo = (ADVAL(n) - poszero)*fattore
```

La procedura calc_xy calcola gli angoli correnti individuati dai due potenziometri e usa questi per definire le distanze x e y del mirino. I concetti di trigonometria alla base del procedimento sono davvero elementari. Con riferimento al diagramma del Braccio, x1 e y1 hanno il seguente rapporto con il Braccio 1:

```
x1 = COS(angolo1)*lungh_Braccio1
y1 = SIN(angolo1)*lungh_Braccio1
```

Il calcolo di x2 e y2 è un po' più complesso, poiché angolo2 è 0 quando il Braccio 2 è allineato al Braccio 1. Comunque la formula è:

```
x2 = COS(angolo1+angolo2)*lungh_Braccio2
y2 = SIN(angolo1+angolo2)*lungh_Braccio2
```

Le distanze totali si ottengono sommando rispettivamente x1 con x2 e y1 con y2.
Le formule usate nel programma sono leggermente diverse poiché, invece delle lunghezze dei bracci, sono adoperate variabili di scala. Queste, oltre a comprendere le dette lunghezze, tengono conto dei fattori di scala necessari per convertire le distanze in coordinate per il sistema grafico del Micro BBC.

Save e Load degli schermi

Il metodo più semplice, per conservare le immagini create sullo schermo, è utilizzare la SAVE per codice macchina, facendo memorizzare la zona di RAM del video usata dal MODE 1. Purché non vi sia stato alcun scorrimento dello schermo dall'ultima esecuzione del comando CLS, quest'area occupa da &3000 a &7FFF. Tuttavia c'è un inconveniente nell'usare il comando *SAVE: il nome del file non può essere passato tramite una variabile stringa. Infatti, il comando:

```
*SAVE file$ 3000 8000
```

memorizza l'area di memoria assegnandole il nome file$, anziché il nome contenuto nella variabile file$. Ciò rende difficile la memorizzazione o la lettura di più schermi, se si intende conservarli sullo stesso nastro o dischetto. Fortunatamente, esiste il modo di aggirare l'ostacolo, adoperando la chiamata al sistema operativo OSCLI. Questo consente di eseguire una serie di comandi depositati sotto forma di stringhe ASCII in qualche punto della memoria, come se si trattasse di comandi impartiti direttamente da tastiera. Per usare tale chiamata occorre passare l'indirizzo iniziale del blocco di caratteri ASCII nei registri X e Y (nel formato basso/alto). Per eseguire la *SAVE e la *LOAD, pertanto, si assembla il relativo comando (completo del nome di file specificato dall'utente) in una stringa, i cui valori ASCII vengono depositati in memoria tramite POKE. Eseguita tale operazione e caricati i registri X e Y con l'indirizzo iniziale del blocco di memoria, si opera la chiamata OSCLI, all'indirizzo &FFF7. Di ciò si occupa la procedura comando_oscli, alla linea 2170.
Oltre alla SAVE e alla LOAD degli schermi, l'operatore può usare il tasto C per ripulire lo schermo e il tasto M per tornare al menu principale. La seconda opzione offerta dal menu principale non funziona ancora, poiché la relativa parte di programma viene presen-

tata in seguito. I tasti 1, 2 e 3 servono per selezionare, in qualsiasi momento, nuovi colori di primo piano. Volendo, i colori abbinati a questi tasti possono essere cambiati tramite il comando VDU 19.

Programma per il tracciatore

La prima metà del programma per il tracciatore, riprodotta qui sotto, consente di usare il Braccio nel modo "a mano libera", poiché le restanti routine (che consentono il tracciamento di singoli punti, di linee e di cerchi) vengono fornite in seguito. Nel modo "a mano libera" sono disponibili sei opzioni. Il pulsante di azionamento, situato sul tracciatore, agisce da controllo per il sollevamento e l'abbassamento della "penna". Questo pulsante è collegato alla porta analogica del BBC come se si trattasse del pulsante di fuoco di un joystick, cosicché il segnale da esso prodotto è rilevabile dal software ispezionando i due bit meno significativi del valore fornito da ADVAL(0). Pertanto, se (ADVAL(0)AND3)<>0 allora il pulsante risulta premuto e va intrapresa l'azione corrispondente. In questo programma, ogni successiva pressione del pulsante fa passare da una condizione all'altra e il fatto è registrato dalla variabile togflag, esaminata nella procedura disegno. In base al valore di togflag viene eseguita una DRAW oppure una MOVE ad una nuova posizione.
Per indicare la posizione del tracciatore, sullo schermo viene usata una piccola crocetta. Questa può spostarsi ovunque, senza disturbare il disegno già tracciato, ricorrendo al modo di tracciamento OR–Esclusivo. Qualsiasi linea tracciata in questo modo (nel colore selezionato da GCOL3) può essere cancellata sovrapponendovi una linea identica, ma di colore uguale allo sfondo. La procedura disegno adopera questa tecnica per cancellare il cursore in una posizione, prima di tracciarlo in un'altra. Le coordinate correnti vengono depositate in oldx e oldy, pronte per essere usate alla successiva chiamata alla procedura disegno.

```
   6 REM ** TRACCIATORE DIGITALE - BBC **
1080 PROCdefiniz_parametri
1090 REPEAT
1100 MODE 1
1110 REM SPENGIMENTO CURSORE
1120 VDU 23,1,0;0;0;0;
1130 PROCmenu
1140 CLS
1150 IF risp$="1" THEN PROCmanolibera ELSE
     PROCelastico
1160 UNTIL flagfine=1
1170 END
```

Questa parte inizializza le variabili, comprese le lunghezze dei bracci e i valori di calibrazione

```
1190 DEF PROCdefiniz_parametri
1200 uno_zero=14820
1210 uno_novanta=45020
1220 uno_lungh=250
1230 fattore_uno=90/(uno_novanta-uno_zero)
1240 due_zero=450
1250 due_novanta=25300
1260 due_lungh=222+28
1270 fattore_due=90/(due_novanta-due_zero)
1280 REM ** FATTORI DI CONVERSIONE DELLE
     COORDINATE **
1290 scala=1023/460
1300 uno_scala=scala*uno_lungh
1310 due_scala=scala*due_lungh
1320 REM ESAMINA SOLO 2 CANALI ADC
1330 *FX 16,2
1340 flagfine=0:flagscelta=0:colore=1
1350 DIM deposito% 30: DIM x(3),y(3)
1360 ENDPROC
1380 DEF PROCmanolibera
1390 PROCinforma_libero
1400 REPEAT
1410 IF (ADVAL(0) DIV 256)<>0 THEN PROCdraw
1420 IF (ADVAL(0) AND 3)<>0 THEN PROCalza_abbassa_penna
1430 risp$=INKEY$(1):IF risp$<>"" THEN PROCpress_libero
1440 UNTIL flaguscita=1
1450 ENDPROC
1470 DEF PROCdraw
1480 PROCcalcola_xy
1490 PROCcursore(vecchiax,vecchiay)
1500 MOVE vecchiax,vecchiay
1510 IF flagscelta=0 THEN DRAW x,y ELSE MOVE x,y
1520 PROCcursore(x,y)
1530 vecchiax=x:vecchiay=y
1540 ENDPROC
1560 DEF PROCcursore(cx,cy)
1570 GCOL 3,3:REM MODO PLOT OR ESCLUSIVO
1580 MOVE cx,cy-16
1590 PLOT 1,0,32
1600 PLOT 0,-16,-16
1610 PLOT 1,32,0
1620 GCOL 0,colore
1630 ENDPROC
```

La seguente procedura esegue i calcoli necessari per convertire le letture fornite dai potenziometri in distanze orizzontali e verticali

```
1650 DEF PROCcalcola_xy
1660 angolo1=RAD((ADVAL(1)-uno_zero)*fattore_uno)
1670 angolo2=RAD((ADVAL(2)-due_zero)*fattore_due)
1680 x=uno_scala*COS(angolo1)+due_scala*COS(angolo1
     +angolo2)
1690 y=1023-(uno_scala*SIN(angolo1)+due_scala*SIN
     (angolo1+angolo2)
1700 ENDPROC
1720 DEF PROCmenu
1730 flaguscita=0:flagfine=0
1740 PRINT TAB(5,10);"Opzioni:"
1750 PRINT TAB(5);"1...Disegno a mano libera"
1760 PRINT TAB(5);"2...Disegno a elastico"
1770 PRINT:PRINT TAB(5);"Premi 1 o 2"
1780 REPEAT:risp$=GET$:UNTIL risp$="1" OR risp$="2"
1790 ENDPROC
1810 DEF PROCalza_abbassa_penna
1820 flagscelta=1-flagscelta
1830 REPEAT UNTIL (ADVAL(0) AND 3)=0
1840 ENDPROC
1860 DEF PROCpress_libero
1870 IF risp$="C" THEN CLS:PROCinforma_libero:
     ENDPROC
1880 IF risp$="M" THEN flaguscita=1
1890 IF risp$="S" THEN PROCsave_schermo:
     PROCinforma_libero:ENDPROC
1900 IF risp$="L" THEN PROCload_schermo:
     PROCinforma_libero:ENDPROC
1910 PROCcambio_colore
1920 ENDPROC
1940 DEF PROCinforma_libero
1950 PROCcalcola_xy:vecchiax=x:vecchiay=y:
     PROCcursore(vecchiax,vecchiay)
1960 PRINT TAB(1,1);SPC(79)
1970 PRINT TAB(1,1);"S=Save  L=Load  M=Menu
     C=Cancella"
1980 GCOL 0,1:MOVE 0,920:DRAW 1280,920
1990 ENDPROC
2010 DEF PROCcambio_colore
2020 risp=VAL(risp$)
2030 IF risp<1 OR risp>3 THEN ENDPROC
2040 colore=risp
2050 ENDPROC
```

Queste procedure consentono di eseguire indirettamente la *SAVE e la *LOAD della porzione di RAM dedicata allo schermo, ricorrendo alla chiamata OSCLI

```
2070 DEF PROCsave_schermo
2080 PROCcursore(x,y):REM CURSORE DISABILITATO
2090 REPEAT
2100 PRINT TAB(1,1);SPC(79)
2110 INPUT TAB(1,1);"SAVE: NOME DEL FILE";file$:
     file$=file$+".S"
2120 UNTIL LEN(file$)<8
2130 file$="*SAVE "+file$+" 3000 8000"
2140 PROCcomando_oscli(file$)
2150 ENDPROC
2170 DEF PROCcomando_oscli(a$)
2180 FOR I=0 TO LEN(a$)-1
2190 deposito%?I=ASC(MID$(a$,I,1,1))
2200 NEXT I
2210 save%?I=13: REM AGGIUNGE RITORNO CARRELLO
2220 X%=deposito% MOD 256: Y%=deposito% DIV 256
2230 CALL &FFF7: REM CHIAMATO OSCLI
2240 ENDPROC
2260 DEF PROCload_schermo
2270 REPEAT
2280 PRINT TAB(1,1);SPC(79)
2290 INPUT TAB(1,1);"LOAD: NOME DEL FILE";file$:
     file$=file$+".S"
2300 UNTIL LEN(file$)<8
2310 file$="*LOAD "+file$
2320 PROCcomando_oscli(file$)
2330 ENDPROC
```

DATABASE

La nona tabella della mappa di memoria del Commodore 64. Per concessione della Commodore Business Machines

INDIRIZZO ESADECIMALE	ETICHETTA	LOCAZIONE DECIMALE	DESCRIZIONE
0299-029A	BAUDOF	665-666	**RS-232 Baud Rate: Full Bit Time (μs)** Porta RS-232, velocità in Baud: intervallo per ogni bit (μs)
029B	RIDBE	667	**RS-232 Index to End of Input Buffer** Porta RS-232: indice per la fine del buffer d'immissione
029C	RIDBS	668	**RS-232 Start of Input Buffer (Page)** Porta RS-232: inizio del buffer d'immissione (valore della pagina)
029D	RODBS	669	**RS-232 Start of Output Buffer (Page)** Porta RS-232: inizio del buffer di uscita (valore della pagina)
029E	RODBE	670	**RS-232 Index to End of Output Buffer** Porta RS-232: indice per la fine del buffer di uscita
029F-02A0	IRQTMP	671-672	**Holds IRQ Vector During Tape I/O** Contiene il vettore IRQ durante le operazioni di I/O su nastro
02A1	ENABL	673	**RS-232 Enables** Abilitazione della porta RS-232
02A2		674	**TOD Sense During Cassette I/O** Segnalatore di TOD durante le operazioni di I/O su nastro
02A3		675	**Temp Storage For Cassette Read** Deposito temporaneo per la lettura da nastro
02A4		676	**Temp D1IRQ Indicator For Cassette Read** Deposito temporaneo per l'indicatore di D1IRQ durante la lettura da nastro
02A5		677	**Temp For Line Index** Deposito temporano per l'indice di linea
02A6		678	**PAL/NTSC Flag, 0 = NTSC, 1 = PAL** Flag PAL/NTSC, 0 = NTSC, 1 = PAL
02A7-02FF		679-767	**Unused** Locazioni non usate
0300-0301	IERROR	768-769	**Vector: Print BASIC Error Message** Vettore: visualizzazione di messaggi d'errore del BASIC
0302-0303	IMAIN	770-771	**Vector: BASIC Warm start** Vettore: partenza "a caldo" del BASIC
0304-0305	ICRNCH	772-773	**Vector: Tokenize BASIC Text** Vettore: conversione in token del testo BASIC
0306-0307	IQPLOP	774-775	**Vector: BASIC Text LIST** Vettore: LIST del testo BASIC
0308-0309	IGONE	776-777	**Vector: BASIC Char. Dispatch** Vettore: emissione di caratteri BASIC
030A-030B	IEVAL	778-770	**Vector: BASIC Token Evaluation** Vettore: valutazione dei token del BASIC

Una banana sensibile

La Banana Interface, così chiamata per il largo impiego di spinotti detti, appunto, "a banana", consente il controllo di dispositivi periferici tramite computer. L'apparecchio fornisce sia uscite sia ingressi per segnali digitali, cosicché è possibile anche progettare sistemi dotati di "feedback"



CRISPIN THOMAS

La Banana Interface

Un'interfaccia per la didattica, l'hobbystica o per qualcosa di più?

La Banana Interface, prodotta dalla Castle Associates (GB), è un apparecchio utile per il controllo di periferiche esterne, progettato appositamente per la didattica. Gli home computer ai quali è collegabile sono il Commodore 64 e il Micro BBC.

Fondamentalmente, l'idea alla base del progetto è simile a quella impiegata nel "Buffer Box", presentato in "Laboratorio" (vedere a pag. 1003), nonostante l'interfaccia Banana sia per molti versi più sofisticata. Intervenendo sulle configurazioni binarie relative ai registri dei dati e della direzione dei dati del computer, i segnali a 5 volt possono essere inviati a dispositivi esterni. Inoltre, poiché le porte per la stampante e per l'utente sono bidirezionali, il computer può anche ricevere segnali dall'esterno, che lo informano sullo stato della periferica sotto controllo (la posizione di un braccio o la pressione di un pulsante, ad esempio). Un'applicazione tipica è quella del controllo di un piccolo robot, combinando opportunamente la direzione dei segnali in ingresso e in uscita.

L'interfaccia viene collegata al computer tramite un cavo piatto al quale sono fissati due connettori, da inserire nella porta per la stampante e per l'utente. L'altra estremità del cavo viene innestata su di un unico connettore a 40 vie, montato sulla stessa interfaccia. L'alimentazione della Banana Interface è duplice; dal cavo collegato al computer proviene una tensione a 5 volt, necessaria per il funzionamento dei circuiti logici per l'invio di segnali lungo le linee dei dati; la seconda tensione, a 12 volt, viene fornita dall'esterno, tramite una coppia di prese "minijack" accanto al connettore a 40 vie.

Sfortunatamente, assieme all'interfaccia non viene venduto il relativo alimentatore. Questa decisione pare che derivi dal fatto che l'apparecchio è principalmente destinato all'uso nelle scuole (non italiane, evidentemente) dove non mancano alimentatori da laboratorio. Comunque per gli appassionati di elettronica non è difficile reperire un alimentatore da 12 volt a 600 milliAmpere. In alternativa, si può adoperare anche un "pacco" di pile o una batteria da auto.

Per realizzare un prodotto veramente adatto ai fini didattici, durante la fase di progetto la Castle Associates si è avvalsa della consulenza di un gruppo di insegnanti e ciò ha influito molto positivamente sul risultato finale.

Il contenitore, realizzato in metallo, è stato progettato tenendo conto delle notevoli sollecitazioni alle quali può essere sottoposto in un ambiente di lavoro quale una scuola.

Anche l'interno è stato progettato mirando ad una lunga e movimentata "vita" dell'apparecchio: i circuiti stampati sono fissati solidamente all'involucro, in modo da evitare falsi contatti a seguito di movimenti bruschi. In sostanza, è veramente difficile mettere fuori uso l'interfaccia.
Sul piano superiore del contenitore si trovano, suddivise in tre gruppi, tutte le prese "a banana" necessarie per predisporre i vari collegamenti. Sulla sinistra si trovano le otto linee d'ingresso e quattro prese di massa. La numerazione di queste prese segue l'ordine binario (1, 2, 4, 8, ecc.), per evidenziare l'abbinamento di ogni linea al valore del bit nel registro d'ingresso. Adiacente a ciascuna presa è collocato un LED, per informare l'utente sul livello logico (1 o 0) della relativa linea.
Il passaggio di un ingresso da un livello logico all'altro viene registrato dal computer e il modo più semplice di verificare ciò è di porre tutte le linee a livello alto. A questo punto, eseguendo una PEEK del registro dei dati, si trova il valore 255, ma connettendo una delle linee a massa tramite gli appositi ponticelli dell'interfaccia, il valore nel registro cambia a seconda della linea posta a livello zero. Questo semplice esperimento sull'interruzione di circuiti può formare la base di un elementare sistema antifurto.

Le linee di uscita

Sulla destra rispetto alle prese di ingresso si trovano quelle di uscita, che occupano gran parte della superficie. Anche in questo caso le prese sono numerate in ordine binario (per facilitare l'abbinamento tra linea e bit del registro) e un LED visualizza il livello di ciascuna linea. Le coppie di prese in uscita sono collegate a circuiti a relais. L'esecuzione di una POKE di un valore nel registro situato all'indirizzo &FE61, causa l'attivazione di uno o più relais, a seconda della combinazione usata.
Collegando quindi un motore con la relativa alimentazione ad una delle prese d'uscita, il suo azionamento può essere comandato via software per mezzo di un programma o di comandi diretti. Alla Banana Interface possono essere collegati fino a 4 motori simultaneamente. Ovviamente, affinché il metodo per controllare motori (ad esempio quelli di una tartaruga o di un piccolo robot) abbia la massima utilità, è bene prevedere una forma di feedback, ossia di ritorno dei segnali, tramite le linee d'ingresso.
Situate sopra le otto linee di uscita appena descritte si trovano altre otto linee di uscita, corrispondenti a otto porte logiche a risposta veloce, capaci di controllare una tensione da 12 volt. Queste linee sono utili per azionare motori passo passo (fino a un massimo di sette alla volta).
Da quanto appena esposto, appare evidente che la Banana Interface è soprattutto un apparecchio destinato alla didattica e alla sperimentazione, anche se il dispositivo può anche essere utilizzato in piccole applicazioni industriali, per automatizzare processi di produzione.
Per i fini hobbystici, però, nonostante l'innegabile utilità sotto il profilo della sperimentazione, l'interfaccia ha un costo tale da scoraggiarne l'acquisto.

BANANA INTERFACE

DIMENSIONI

mm. 300 x 200 x 63

INTERFACCE

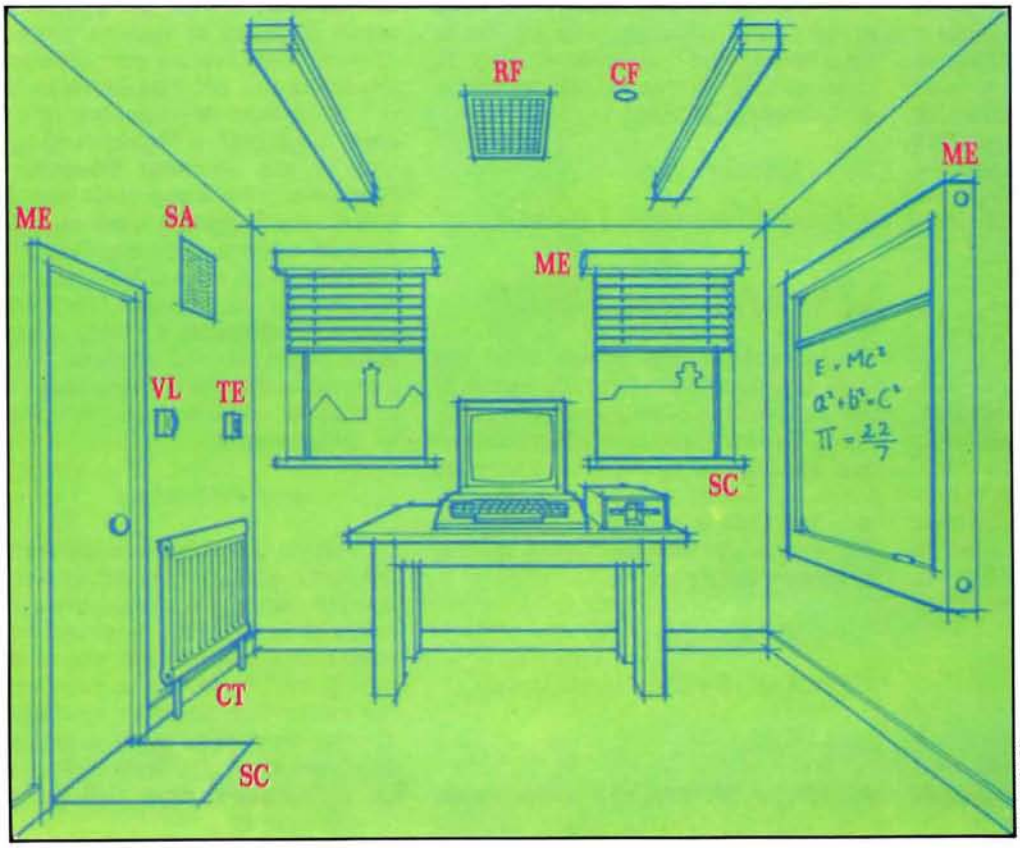
Connettore parallelo a 40 vie, da collegare alla porta utente e per la stampante del computer (Micro BBC o Commodore 64). Otto prese logiche di ingresso e otto di uscita, più otto ad alta velocità

PRO

Ben realizzato, robusto e adatto ad un'ampia serie di applicazioni didattiche. In particolari casi può essere adoperato per piccole applicazioni commerciali o industriali

CONTRO

Per l'hobbysta l'acquisto dell'apparecchio (sprovvisto di alimentatore a 12 volt necessario per il funzionamento) è certamente oneroso. La Banana Interface, infatti, è più adatta come strumento didattico



La figura mostra una possibile applicazione della Banana Interface nell'ambito domestico o scolastico, per controllare tutta una serie di dispositivi tramite computer. Sensori di vario tipo, collocati opportunamente nella stanza, inviano segnali all'interfaccia, che li trasmette al computer. Questo, a sua volta, esamina i segnali ricevuti e li confronta con i rispettivi valori ottimali. Se uno dei segnali esce dai parametri prestabiliti (ad esempio la temperatura ambientale scende al di sotto di un certo valore), il software interviene emettendo un segnale che, sempre tramite l'interfaccia, agisce sul dispositivo di regolazione

SC = Sensori di Contatto
CT = Controllo Termico
VL = Variatore di Luminosità
ME = Motore Elettrico
SA = Sirena d'Allarme
RF = Rivelatore di Fumo
TE = Termostato
CF = Cellula Fotosensibile

Dichiarazioni d'interesse

Un esame della natura dichiarativa del linguaggio PROLOG, corredato da esempi

Il PROLOG è stato progettato per offrire al programmatore un mezzo per descrivere la struttura logica di un problema usando una specie di archivio di "fatti" e regole condizionali (se...allora...). La frase "dichiarativa" del problema viene usata dai meccanismi interni del PROLOG per fornire risposte ai quesiti formulati dall'utente. Il ruolo del programmatore, dunque, non è più quello di informare il computer, con grande profusione di particolari, su come usare le operazioni espresse in un linguaggio di programmazione, bensì quello di fornire una chiara e logica dichiarazione sulla conoscenza necessaria per risolvere una particolare serie di problemi, lasciando poi al PROLOG la fatica di trovare le soluzioni.
La sintassi del PROLOG è molto semplice. Al contrario, la terminologia adottata è, in un primo momento, molto difficile da apprendere. Il costrutto fondamentale è l'"elemento". Un elemento può essere una "costante" (ad esempio: "albero", "Marco", "25"), una "variabile" oppure una "struttura". Le strutture vengono create usando costanti e variabili e una struttura abbastanza comune, nei programmi in PROLOG, è nota col termine "fatto". Questo consiste in un predicato (che deve essere una costante) a sé stante oppure seguito da una lista di argomenti (che possono essere costanti o variabili) racchiusi tra parentesi.
Ad esempio:

```
predicato1.
predicato2(argomento1, argomento2,...).
```

Come già affermato in precedenza, un predicato può essere considerato come un rapporto esistente tra i vari elementi che formano gli argomenti. Nella frase «Marco mangia ciliegie», "mangia" è il predicato che abbina Marco alle ciliege.
In PROLOG, questo fatto si esprime con:

```
mangia(Marco, ciliegie).
```

Elementi come questo possono essere raggruppati traendo spunto da una vasta serie di fatti, così come singoli record vengono accumulati in un grosso database. Si supponga di aver stilato il seguente ipotetico database di fatti concernenti le popolazioni planetarie e le loro abitudini alimentari:

```
mangia(venusiano, alberi).
mangia(gioviano, rocce).
mangia(terrestre, pane).
mangia(mercuriano, vermi).
mangia(nettuniano, pane).
mangia(marziano, marziani).
```

Senza aggiungere altro codice, questo elenco rappresenta già un programma in PROLOG, ma per utilizzarlo occorre impostare una proposizione che il PROLOG provvede a verificare.
Volendo sapere, ad esempio, se i gioviani mangiano rocce, si imposta il quesito scrivendo:

```
?— mangia(gioviano, rocce).
```

dopo il prompt del sistema (generalmente ?—). Il PROLOG esamina i vari fatti nel database e, se trova una proposizione corrispondente in tutte le sue parti risponde con un Yes, a significare che il quesito è vero, in base ai fatti contenuti nel database. Se adesso si chiede:

```
?— mangia(marziano, pane).
```

la risposta risultante è un No.

Le variabili del PROLOG

Il programma può essere reso più utile introducendo l'uso di variabili nei quesiti. Per convenzione, in PROLOG, i nomi delle variabili iniziano con una lettera maiuscola, mentre le costanti iniziano sempre con una lettera minuscola.
Per sapere chi mangia vermi, quindi, possiamo scrivere:

```
?— mangia(Creatura, vermi).
```

Il PROLOG fornisce la risposta:

```
Creatura = mercuriano
```

poiché ha trovato che assegnando alla variabile Creatura il valore mercuriano è in grado di provare come vera la proposizione. A questo punto il PROLOG resta in attesa di un'immissione: si può premere Return, se si è soddisfatti della risposta, oppure un punto e virgola per chiedere al PROLOG di trovare un'ulteriore corrispondenza. In questo caso non esistono altre proposizioni che risultino vere, quindi la risposta è No. Se invece si chiede chi mangia pane, esistono due possibili risposte: terrestre e nettuniano, cosicché al quesito:

```
?— mangia(Creatura, pane).
```

il PROLOG risponde con:

```
Creatura = terrestre
```

e, se ora si digita un ; il PROLOG prosegue con:

```
Creatura = nettuniano
```

Se esistessero altre soluzioni, esse verrebbero visualizzate con ulteriori pressioni del punto e virgola. Il funzionamento di questo meccanismo viene spiegato in seguito: prima occorre soffermarsi su altri tre concetti fondamentali del PROLOG: le "regole", le "catene di inferenza" e i "percorsi a ritroso" o "backtracking".
Uno o più elementi formano una "clausola", che a sua volta può esprimere una "regola". Una clausola è sempre composta da una "testa" (che consiste di un elemento) e può essere seguita da un "corpo", formato da uno o più elementi. La testa e il corpo sono separati dal simbolo :—, che generalmente ha il significato della particella condizionale "se". Pertanto, il formato è:

```
elem1 :— elem2, elem3, elem4.
```

Le virgole che separano gli elementi del corpo sono paragonabili alla condizione logica AND (congiunzione "e"), cosicché la precedente struttura può essere così interpretata: elem1 è vero SE elem2 E elem3 E elem4 sono tutti veri.
Per chiarire il concetto, si supponga di voler trovare se tra le creature nel database esistono degli antropofagi. La definizione di cannibale può essere espressa con:

```
cannibale(Creatura):— mangia(Creatura, Creatura).
```

Questa è una clausola con un solo elemento nel corpo ed è interpretabile come una regola che recita: "una creatura è cannibale se tale creatura mangia lo stesso tipo di creatura".
Dopo aver aggiunto questa regola al programma, si può chiedere:

```
?— cannibale(X).
```

Il PROLOG abbina questa proposizione con la testa della nuova clausola e, poiché quest'ultima non è automaticamente vera, va prima controllata la veridicità degli elementi che compongono il corpo della clausola. Il PROLOG, quindi, prende ciascun elemento a turno e costruisce una proposizione da verificare, come se questa venisse di volta in volta digitata alla tastiera. Nel caso in questione esiste un solo elemento nel corpo, quindi la proposizione risultante diventa:

```
..., mangia(Creatura, Creatura).
```

che corrisponde, in effetti, a uno dei fatti contenuti nel database: mangia(marziano, marziano). Alla variabile Creatura viene quindi assegnato il valore marziano. Di conseguenza può essere verificata la proposizione cannibale(marziano), assegnando alla variabile X, usata nel quesito, il valore marziano. La risposta del PROLOG è dunque:

```
X = marziano
```

Tutto ciò avviene perché nel database è stato trovato il fatto mangia(marziano, marziano). Ma, se invece di trovare un semplice fatto, si fosse trattato di un'altra regola, il PROLOG avrebbe dovuto costruire un'ulteriore proposizione e proseguire nella ricerca.
Il seguente programma in PROLOG dimostra quanto appena affermato:

```
colore_di(Marziano, rosa):— umore_di(Marziano,
    buono)
colore_di(Marziano, blu)
umore_di(Marziano, buono):—
programma_in(Marziano, prolog).
umore_di(Marziano, triste).
programma_in(marzio, basic).
programma_in(miranda, prolog)
```

Se, per trovare il colore di marzio, si pone il quesito:

```
?— colore_di(marzio, Colore).
```

il PROLOG prima considera il quesito come proposizione, poi trova una regola per decidere se il colore_di(marzio, rosa) è vero o falso e scopre che è vero se (:—) l'umore di marzio è buono. Il PROLOG considera quindi questa come la successiva proposizione e scopre adesso una regola che stabilisce: "marzio è di buon umore se programma in PROLOG". Quindi programma_in(marzio, prolog) diviene la successiva proposizione.
Il meccanismo può proseguire per un gran numero di livelli ma, in questo caso, non esiste alcuna regola o fatto che mostri come provare che marzio è un programmatore in PROLOG.
Tuttavia, il programma non si arresta di fronte a questo ostacolo: ammette semplicemente di aver fallito con la proposizione corrente e torna indietro a quella precedente per vedere se esiste una soluzione alternativa. Anche questa fallisce (non si può provare che marzio è di buon umore), pertanto il PROLOG esegue un nuovo "percorso a ritroso" (backtrack) per tentare, a questo punto, di dimostrare la prima proposizione. Qui, in effetti, il PROLOG trova che colore_di(Marziano, blu) assegna a Marziano il valore marzio e la proposizione può essere provata.
Grazie a questo processo di ricerca, esaminando a turno i vari livelli e creando nuove proposizioni, il PROLOG riesce a rispondere a tutti i quesiti formulati dall'operatore. Il "percorso a ritroso" applicato durante la ricerca assomiglia al metodo spesso usato nell'Intelligenza Artificiale per trovare l'uscita da un labirinto.

IL PROGRAMMA ALVEY

Il linguaggio PROLOG ha assunto un ruolo predominante nel programma di ricerca Alvey, per lo sviluppo dei "supercomputer" della Quinta Generazione. Il progetto è iniziato nel 1983 a seguito del rapporto della Commissione Alvey, un organismo creato per offrire consulenza e collaborazione nella ricerca, in risposta alle iniziative del Giappone circa il proprio progetto per computer della Quinta Generazione.
La collaborazione è l'elemento chiave del progetto Alvey, che si affianca al Progetto Europeo per la Quinta Generazione (ESPRIT), anche nella formulazione di intese con i ricercatori nipponici.
Una delle aree di ricerca prevede la realizzazione di una macchina PROLOG e gli studi iniziali sembrano confermare che sia possibile realizzare apparecchi sfruttando le acquisizioni tecnologiche già esistenti. Ciò significa che questo tipo di computer, sul quale è facilissimo implementare i cosiddetti "sistemi esperti", può essere commercializzato a prezzi accettabili (minori di quelli per i minicomputer) ed in un tempo relativamente breve.
Un'altra area di ricerca esplora l'uso del PROLOG come interfaccia uomo/macchina in linguaggio naturale. Questo è, tra l'altro, uno degli obiettivi di punta del progetto per la Quinta Generazione



KOWALSKI: per concessione della COMPUTER WEEKLY

Il progenitore del PROLOG

Il professor Bob Kowalski è spesso chiamato il "padre della programmazione logica". Le sue intuizioni sulla programmazione e sui sistemi logici hanno fortemente influenzato A. Colmerauer nello sviluppo del primo interprete PROLOG, presso l'Università di Marsiglia

Profitti e perdite

Nella parte conclusiva del gioco di simulazione "Nuovo Mondo" si tirano le somme

Comincia un nuovo giorno, dopo che le trattative commerciali si sono concluse e le merci acquistate, assieme alle armi rimaste, sono state caricate sulla nave. L'equipaggio provvede a rifornirsi di acqua fresca e di provviste alimentari per prepararsi al viaggio di ritorno in patria, dove le perle, i manufatti e le spezie verranno rivendute con buon profitto (o almeno così tutti sperano).
Durante la notte precedente alla partenza, però, si verifica un fatto inatteso: un rivale del capo indigeno visita segretamente la nave ed offre ben 30 perle per ogni fucile a bordo. I suoi intenti, nei confronti dei propri compaesani, non sono decisamente pacifici. Il guadagno proveniente da questa trattativa poco ortodossa è allettante ma (scrupoli a parte) vanno fatte alcune considerazioni.
Se il capo indigeno scopre il complotto, venendo così a sapere della vendita, è inevitabile una rappresaglia nei confronti della nave, che viene incendiata facendo terminare il gioco in modo poco onorevole. Tuttavia, se si pensa che il margine di guadagno finora ottenuto sia scarso, può valere la pena di correre il rischio.
Ovviamente, la richiesta del rivoltoso può essere rifiutata, nel qual caso il capo indigeno, per dimostrare la propria gratitudine, fa omaggio di scorte per il viaggio di ritorno e può, addirittura, arrivare a regalare anche 50 perle.
Comunque, se tutto va per il verso giusto, la nave salpa con il vento in poppa e l'equipaggio in perfetta forma. Durata prevista della traversata: 8 settimane. Giunti a destinazione, le merci acquistate vengono rivendute, all'equipaggio viene consegnata la paga e, probabilmente, rimane anche un buon margine di guadagno. Se il denaro per le paghe dei marinai non dovesse bastare, può sempre essere venduta la nave. Il rapporto finale presentato dal programma fornisce la situazione contabile della spedizione, informando se il viaggio ha prodotto un profitto o una perdita.
Il programma principale, alla linea 893, richiama la subroutine alla linea 10300, che si occupa dell'offerta di acquisto delle armi da parte del rivoltoso. Per prima cosa, la subroutine verifica che a bordo esistano fucili, altrimenti le trattative non avrebbero alcun senso. La linea 10305 esamina il secondo elemento della matrice delle scorte, OA(2), per vedere se ci sono armi. Se il valore è 0, ossia di fucili non c'è nemmeno l'ombra, il controllo torna al programma principale. In caso contrario il rivoltoso annuncia la sua offerta.
L'accettazione o il rifiuto di tale offerta da parte del capitano della nave è immesso alla linea 10326, digitando una S o una N. La linea 10330 verifica la validità della risposta, ripetendo la domanda se c'è un errore.
Se si accetta la proposta, la routine salta alla linea 10400, che inizia le trattative. La linea 10405 genera un numero casuale tra 0 e 1, fornendo un 75% di probabilità che il programma salti alla linea 10450, che informa il giocatore dell'avvenuta scoperta del complotto da parte del capo indigeno. Se si ritiene che tale percentuale di probabilità sia troppo alta, la si può variare a piacimento.
Se la rivolta fallisce, il capo indigeno si riprende tutte le merci dalla nave, che viene incendiata, e il gioco termina miseramente. Se invece la rivolta ha successo il programma prosegue con la linea 10429: il nuovo capo, in segno di riconoscenza, offre doni al capitano della nave.

Basic & Basic

Spectrum:

Sostituire AO( ) con E( ), V1( ) con B( ) e V2( ) con D( ) in tutto il programma e apportare le seguenti modifiche:

```
 1310 CLS: GO SUB 9200
10328 INPUT I$:LET I$=I$(TO 1)
10354 LET I$=INKEY$: IF I$="" THEN GO TO 10354
10400 CLS: GO SUB 9200
10501 CLS: GO SUB 9200
10547 LET I$=INKEY$:IF I$="" THEN GO TO 10547
```

Micro BBC:

Apportare le seguenti modifiche:

```
10310 CLS: GO SUB 9200
10354 I$=GET$
10400 CLS: GOSUB 9200
10501 CLS: GOSUB 9200
10574 I$=GET$
```

Aggiornamento delle matrici

L'espressione alla linea 10429 calcola il numero di perle ottenute nello scambio, moltiplicando il numero dei fucili, OA(2), per 30. Il risultato viene sommato al valore contenuto nel primo elemento di AO( ), che contiene il totale di perle acquistato. La linea 10430, infine, elimina la presenza delle armi dalla matrice delle scorte.
Quest'ultima operazione, ai fini del gioco, non è strettamente necessaria, ma può essere utile se si desidera ampliare il gioco aggiungendo nuovi eventi durante il viaggio di ritorno.
Se il giocatore rifiuta l'offerta del rivoltoso, il capo indigeno può regalare fino a 50 perle in segno di riconoscenza.
La quantità di perle è stabilita in modo

MODULO 13: IL VIAGGIO TERMINA

Aggiunte al programma principale

```
893 GOSUB 10300
894 GOSUB 10500
```

Routine della rivolta

```
10300 REM RIVOLTA
10305 IF OA(2)=0 THEN RETURN
10310 PRINTCHR$(147):GOSUB9200
10315 S$="DURANTE LA NOTTE UN RIVALE DEL*":
      GOSUB9100
10316 S$="CAPO VISITA IN SEGRETO LA NAVE*":
      GOSUB9100
10317 PRINT:GOSUB9200
10318 S$="DESIDERA ACQUISTARE ARMI PER*":
      GOSUB9100
10320 S$="UNA RIVOLTA*": GOSUB9100
10322 PRINT:  GOSUB9200
10324 S$="OFFRE 30 PERLE PER OGNI ARMA*": GOSUB9100
10326 S$="VUOI VENDERGLI ARMI (S/N)*": GOSUB9100
10328 INPUT I$:I$=LEFT$(I$,1)
10330 IF I$<>"N" AND I$<>"S" THEN 10328
10332 IFI$="S" THEN 10400
10334 PRINT: GOSUB9200
10336 S$="IL CAPO HA SAPUTO ED E'*": GOSUB9100
10338 S$="RICONOSCENTE: TI REGALA DELLE*":
      GOSUB 9100
10340 S$="UTILI SCORTE PER IL *": GOSUB9100
10342 S$="VIAGGIO DI RITORNO*": GOSUB9100
10344 GOSUB9200
10345 IF RND(1)<.75 THEN 10350
10346 S$="E ANCHE 50 PERLE!!*": GOSUB9100
10348 AO(1)=AO(1)+50
10350 PRINT: GOSUB9200
10352 S$=K$: GOSUB9100
10354 GETI$: IFI$=""THEN10354
10359 RETURN
10400 PRINTCHR$(147): GOSUB9200
10405 IFRND(1)<.75 THEN 10450
10410 S$="LA RIVOLTA HA SUCCESSO!*": GOSUB9100
10412 PRINT: GOSUB9200
10415 S$="IL NUOVO CAPO TI REGALA*": GOSUB9100
10420 S$="UTILI SCORTE PER IL VIAGGIO*": GOSUB9100
10425 S$="DI RITORNO.*":GOSUB9100
10429 AO(1)=AO(1)+(OA(2)*30): REM AGGIUNGE LE
      PERLE
10430 OA(2)=0
10431 GOTO10350
10450 S$="LA RIVOLTA FALLISCE!!*": GOSUB9100
10452 PRINT: GOSUB9200
10455 S$="IL VECCHIO CAPO E' ADIRATO:*":
      GOSUB9100
10457 S$="FA BRUCIARE LA TUA NAVE E*": GOSUB9100
10458 S$="SACCHEGGIA TUTTO!!*": GOSUB9100
10459 PRINT: GOSUB9200
10460 S$=" FINE DEL GIOCO!!*": GOSUB9100
10462 END
10464 GOTO10462
```

Routine di fine viaggio

```
10500 REM CONCLUSIONE DEL VIAGGIO
10501 PRINTCHR$(147): GOSUB9200
10505 S$="CON L'EQUIPAGGIO IN FORMA E*": GOSUB9100
10507 S$="VENTI FAVOREVOLI IL RITORNO*": GOSUB9100
10508 S$="DURA SOLTANTO 8 SETTIMANE.*": GOSUB9100
10512 WW=0
10514 FOR T=1 TO 5
10516 WW=WW+(8*CC(T)*WG(T))
10518 NEXT
10519 PRINT: GOSUB9200
10520 S$="SALARI PER IL RITORNO = *": GOSUB9100
10522 PRINTWW;"MONETE D'ORO"
10524 PRINT: GOSUB9200
10526 S$="ALL'ARRIVO IN PORTO*":GOSUB9100
10528 S$="VENDI LE MERCI IL CUI VALORE*":
      GOSUB9100
10530 S$="ATTUALE E':*": GOSUB9100
10532 PRINT"PERLE - ";V2(1);"MONETE D'ORO"
10534 PRINT"MANUFATTI - ";V2(2);"MONETE D'ORO"
10536 PRINT"SPEZIE - ";V2(3);"MONETE D'ORO"
10538 PRINT:S$="PER UN TOTALE DI *": GOSUB9100
10540 X=(AO(1)*V2(1))+(AO(2)*V2(2))+(AO(3)*V2(3))
10542 PRINTX;"MONETE D'ORO"
10545 PRINT:S$=K$: GOSUB9100: PRINT
10547 GET I$:IF I$="" THEN 10547
10550 S$="ADESSO POSSIEDI:*": GOSUB9100
10552 PRINTMO+X;"MONETE D'ORO"
10555 PRINT: GOSUB9200
10556 S$="SALARI PER IL RITORNO:*": GOSUB9100
10577 PRINTWT+WW;"MONETE D'ORO"
10559 PRINT: GOSUB9200
10560 S$="IL GIOCO TERMINA CON:*": GOSUB9100
10562 Z=MO+X-WT-WW
10565 PRINTZ;"MONETE D'ORO"
10566 PRINT:GOSUB9200
10567 PRINT:S$="SEI NOMINATO *": GOSUB9100: PRINT
10568 IF Z>3200 THEN S$="GRAN FINANZIERE*": GOSUB
      9100: END
10569 IF Z>2500 THEN S$="MAESTRO COMMERCIANTE*":
      GOSUB9100: END
10570 IF Z>2000 THEN S$="MERCANTE DI TERZA
      CLASSE*": GOSUB9100: END
10571 IF Z>1000 THEN S$="... UN VERO POLLO!*":
      GOSUB9100: END
10572 S$="DISASTRO DELL'ANNO!*"
10573 GOSUB9100: END
```

casuale dalla linea 10345. La routine conclusiva del gioco (il ritorno e la rivendita delle merci) inizia alla linea 10500.

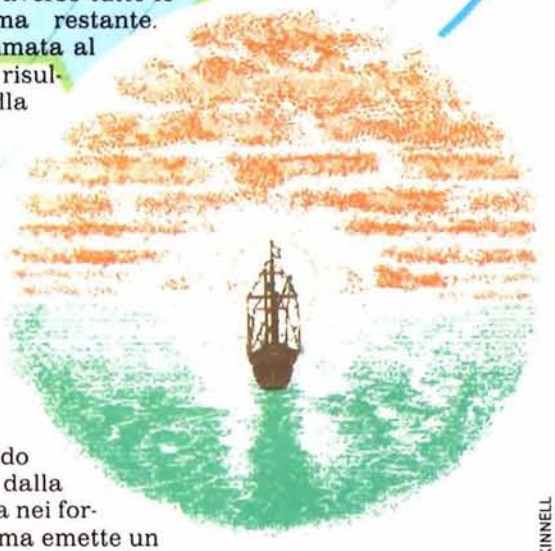
Anche durante la traversata di ritorno, che dura 8 settimane, vanno calcolate le paghe dell'equipaggio: a tale scopo, alla linea 10512, viene creata la variabile WW. Il suo valore è calcolato all'interno di un ciclo che inizia alla linea 10514. La formula moltiplica il numero di ciascun tipo di membri dell'equipaggio, CC(T), per la corrispondente paga settimanale WG(T). Il contatore T garantisce che il calcolo comprenda tutti i membri dell'equipaggio, producendo così il totale settimanale delle paghe dovute. Questo valore, moltiplicato per 8 (il numero delle settimane di viaggio), viene quindi visualizzato.

Quando la nave raggiunge il porto, le merci vengono rivendute. I relativi prezzi di mercato, stabiliti in modo casuale all'inizio del viaggio, sono contenuti nella matrice V2(3), dimensionata alla linea 62. Le linee da 10532 a 10536 visualizzano tali valori per ciascun tipo di merce.

L'ammontare totale dell'oro ricavato dalla vendita delle merci è calcolato dalla linea 10540 e depositato in X. La formula impiegata non fa che moltiplicare la quantità di ciascuna merce, conservata in AO(), per il prezzo di mercato corrente, in V2(). Viene quindi visualizzato il totale delle entrate risultanti dalla vendita delle perle, dei manufatti e delle spezie.

Può darsi che non tutto il capitale iniziale (le 2000 monete d'oro) sia stato speso: la variabile MO ha contenuto, attraverso tutte le vicissitudini, la somma restante. Questa cifra viene sommata al totale delle entrate e il risultato è visualizzato dalla linea 10552.

Il totale dovuto per le paghe è calcolato alla linea 10557 sommando semplicemente le paghe dovute per l'andata, contenute in WT, a quelle dovute per il ritorno. Il programma calcola quindi l'eventuale profitto di tutta la missione, sottraendo il totale delle spese dalla somma in oro contenuta nei forzieri. Infine, il programma emette un commento sulla bravura del giocatore, commisurato al margine di profitti ottenuto.



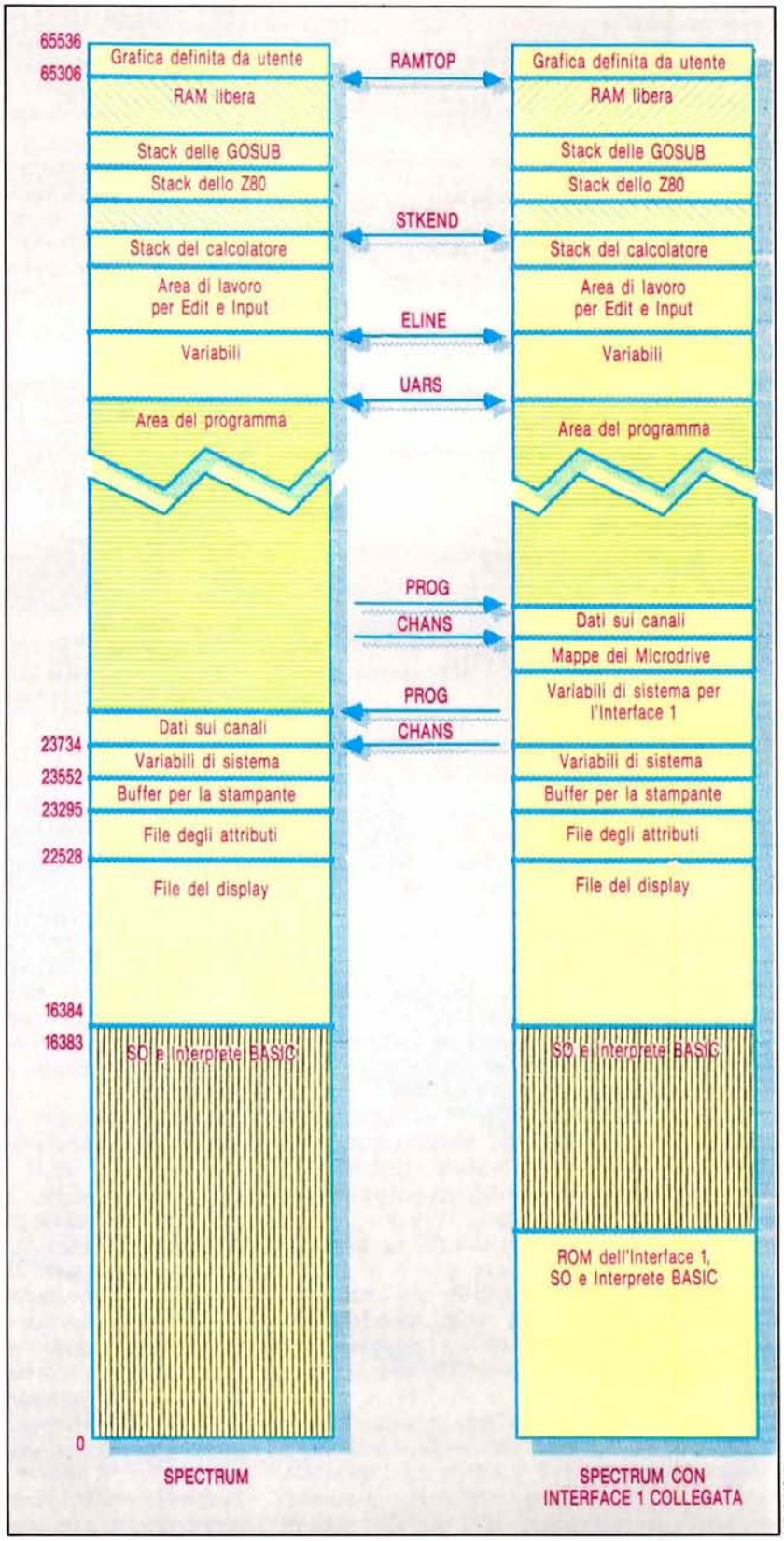
IAN McKINNELL

I segreti dello Spectrum

La prima parte di una serie dedicata al sistema operativo dello Spectrum

Lo Spectrum della Sinclair, a differenza del Micro BBC, non possiede una ROM esclusivamente dedicata al sistema operativo: l'interprete BASIC e il SO sono sullo stesso chip. Inoltre, è sempre esistita una sola versione di questa ROM, tant'è vero che è stata utilizzata anche per lo Spectrum+.
Il SO è una compatta serie di routine, che occupa soltanto 7 Kbyte dei 16 disponibili sul chip di ROM. Le dimensioni del SO per lo Spectrum sono circa la metà di quelle per il Micro BBC.
Nonostante esista un'unica versione del sistema operativo e il codice che lo compone sia estremamente compatto, c'è una differenza sostanziale con il SO del Micro BBC: quest'ultimo è vettorizzato mentre quello dello Spectrum, a parte qualche piccola eccezione, no. Ciò significa che è meno facile modificare il comportamento delle routine, aggiungendone delle nuove. Tuttavia, l'ostacolo è aggirabile, come si avrà modo di scoprire più avanti.
Chi abbia seguito la serie dedicata al sistema operativo del Micro BBC (vedere a pag. 1338) è già a conoscenza del ruolo assunto dal SO in un computer. In breve, questo ruolo può essere descritto come la gestione di tutte le routine riguardanti l'ingresso/uscita dal sistema, la visualizzazione, l'archiviazione su file e il trattamento delle interrupt. In sostanza si tratta di un'interfaccia tra l'hardware e i programmi dell'utente o l'interprete del BASIC. L'esame del SO per lo Spectrum inizia con un'analisi della mappa di memoria, mostrata qui a fianco.
Il primo blocco di 16 Kbyte di memoria è occupato dal chip di ROM che contiene il sistema operativo e l'interprete BASIC. Grosso modo, il SO occupa 7–8 Kbyte e l'interprete la parte restante. Quando viene collegata l'Interface 1, gli 8 Kbyte più bassi vengono impaginati in modo simile a quello usato sul Micro BBC per la zona di memoria tra &8000 e &BFFF. In pratica è come se questi 8 Kbyte venissero sostituiti fisicamente con un nuovo chip contenente le routine per i Microdrive, per l'interfaccia seriale e per il sistema a Rete Locale (LAN). Questo aspetto viene discusso più avanti.
Il resto della memoria è di tipo RAM, ma la

Con e senza Interface 1
I diagrammi mostrano il frazionamento della memoria RAM di uno Spectrum da 48 Kbyte (valido anche per lo Spectrum+), a seconda che sia collegata o no l'Interface 1. L'area riservata alle variabili di sistema inizia in genere da 23733, ma la sua estensione aumenta quando viene collegata l'Interface 1



sua utilizzazione, come si può vedere dal diagramma, varia a seconda che l'Interface 1 sia collegata o no.
Qui di seguito viene data una descrizione delle singole zone di memoria.
Il *file del display* contiene le informazioni concernenti la forma delle immagini riprodotte sullo schermo. I dati relativi al colore e alle condizioni BRIGHT e FLASH non sono contenute in questa zona, bensì nell'apposito *file degli attributi*, poiché tali informazioni si riferiscono a intere celle di carattere sullo schermo.
Il *buffer di stampa* è un'area di memoria lunga 256 byte, usata dalla stampante ZX quando viene richiesto uno stampato.
Le *variabili di sistema* sono usate dal SO e dall'interprete per controllare cosa sta succedendo in ogni momento nel sistema. Alcune variabili di sistema (il cui elenco completo è riprodotto sul manuale dello Spectrum ma, stranamente, non su quello dello Spectrum+) vengono analizzate più attentamente in seguito.
Questa è la zona di memoria maggiormente influenzata dalla presenza dell'Interface 1, che richiede l'impiego di un maggior numero di variabili di sistema (situate subito dopo quelle normali) e di aree di lavoro. Questa maggiore necessità di spazio, ovviamente, provoca uno scorrimento verso l'alto di tutte le successive zone di memoria, implicando alcuni aggiustamenti.
Anche le *mappe dei Microdrive* sono caratteristiche relative all'Interface 1 e servono per sapere quali sezioni del nastro sono occupate nel Microdrive.
L'*area dei dati sui canali* consente un modo di vettorizzare, anche se limitato, i comandi LLIST, LPRINT, INPUT# e PRINT# del BASIC. Ad esempio, si può scrivere una routine in codice macchina, che viene utilizzata al posto di quella originale, per alterare i comandi diretti alla stampante ad ogni esecuzione di una LPRINT o LLIST. Un esempio di applicazione simile viene fornito più avanti. L'area dei dati sui canali viene utilizzata anche dall'Interface 1.
A questo punto s'incontra l'area di memoria riservata al programma BASIC e alla relativa zona di lavoro, usata per conservare il testo del programma e le variabili da esso usate.
Le zone di lavoro per l'edit e l'input sono adoperate dal sistema, rispettivamente, per la manipolazione dei caratteri immessi da tastiera e per l'immissione di linee di programma. Appena terminata l'immissione, il testo viene aggiunto a quello del programma e le zone vengono liberate.
Sopra queste zone si trova un gruppo di stack. Lo *stack di calcolo* è quello adoperato dall'interprete BASIC per le operazioni matematiche. Lo *stack dello Z80* è usato per le operazioni della CPU (conservazione temporanea degli indirizzi di ritorno delle routine o di particolari registri interni). Lo *stack delle* GOSUB è usato dall'interprete BASIC per conservare l'indirizzo immediatamente successivo a quello di una chiamata GOSUB. Infine, l'*area per la grafica dell'utente* contiene, come è facile intuire, le definizioni dei caratteri definiti dall'utente.

Dove collocare nuove routine

Esaminando la mappa di memoria appare subito evidente che non esiste una zona libera "ufficiale" dove inserire routine in codice macchina scritte dall'utente. Comunque, ci sono tre aree dove è possibile un simile intervento. Una è nella linea 1 REM di un programma BASIC: la posizione in memoria di tale linea è facilmente desumibile dal valore della variabile di sistema PROG, alla quale basta sommare 5 per avere l'indirizzo del primo byte nella linea.
Le altre due zone sono: il buffer della stampante (purché quest'ultima non venga usata) e l'area che va da RAMTOP all'inizio della zona riservata ai caratteri definiti dall'utente. L'utilizzo di queste zone è riassunto in questa tabella:

Zona	Spectrum	Spectrum e Interface 1
Linea 1 REM	SI	NO
Buffer stampante	SI(*)	SI(*)
Sopra RAMTOP	SI	SI

(*) Da non adoperare se la stampante è in uso

Come si può vedere, la scelta più sicura, in qualsiasi circostanza, è l'ultima. Il codice collocato sopra RAMTOP è automaticamente protetto da eventuali sovrascritture da parte del BASIC e riservare spazio sufficiente è molto semplice. L'esatta posizione di RAMTOP è contenuta nelle locazioni di memoria 23730 e 23731, nel formato consueto per lo Z80, ossia byte basso/byte alto. Per risalire alla posizione corrente di RAMTOP, quindi, si usi:

```
LET ramtop = PEEK23730 + 256*PEEK23731
```

Ottenuto e annotato tale valore (che a memoria libera, su uno Spectrum da 48 Kbyte è generalmente 65306), occorre stabilire quanti byte occorrono per la routine in codice macchina da inserire e calcolare il nuovo valore da assegnare a RAMTOP, con la formula: valore = RAMTOP–(lunghezza in byte del codice macchina + 1). Il valore di RAMTOP va modificato poi con:

```
CLEAR valore
```

Va notato che l'esecuzione di questo comando all'interno di un programma BASIC provoca un CLS, ripristina la posizione iniziale di PLOT e opera una RESTORE. Inoltre, azzera e ripristina lo stack delle GOSUB al valore iniziale. Considerate queste conseguenze, è bene scegliere attentamente dove collocare il comando CLEAR in un programma. Preferibilmente, esso va posto all'inizio, per non compromettere il funzionamento delle operazioni eseguite in precedenza.
Eseguito il comando CLEAR valore, il nuovo indirizzo di RAMTOP viene depositato automaticamente nelle apposite locazioni e l'indirizzo

del primo byte disponibile per il codice macchina è dato da: valore + 1. Pertanto, supponendo di avere la memoria completamente libera in uno Spectrum da 48 Kbyte, il comando CLEAR 59999 riserva circa 5 Kbyte di memoria per i programmi in codice macchina, a partire dalla locazione 60000. Una volta riservata la memoria, il codice macchina può essere tranquillamente depositato tramite il comando POKE.

Alcune variabili di sistema

Questa parte introduttiva al SO dello Spectrum si conclude con la descrizione di alcune variabili di sistema, utili per conoscere come venga assegnata la memoria del sistema.
L'accesso a dette variabili avviene adoperando i comandi PEEK e POKE (non esiste alcun equivalente alle chiamate OSBYTE e OSWORD del BBC, per intendersi).

- chars: questa variabile, situata in 23606 e 23607, punta all'inizio di una zona di 256 byte di memoria subito sotto le informazioni sul set dei caratteri, ossia 256 sotto alla definizione del carattere "spazio" (CHR$32). L'effettivo indirizzo si trova con:

```
LET chars = PEEK 23606 + 256 * PEEK23607
```

Volendo si può depositare un nuovo indirizzo in queste locazioni, per utilizzare, ad esempio, un nuovo set di caratteri in RAM.

- vars: questa variabile, che si trova nelle locazioni 23627 e 23628, contiene l'indirizzo iniziale delle variabili BASIC in memoria. Il valore di questa variabile può essere esaminato con PEEK, ma non lo si modifichi, perché altrimenti lo Spectrum finisce col confondersi!

- prog: questa è la variabile di sistema che contiene l'indirizzo iniziale del programma BASIC ed è situata nelle locazioni 23635 e 23636. L'indirizzo del primo byte di un programma BASIC si può dunque ricavare con:

```
LET prog = PEEK23635 + 256 * PEEK23636
```

calcolando anche vars (vedi sopra), allora:

```
PRINT vars—prog
```

visualizza la lunghezza in byte del programma BASIC in memoria.

- eline: questa variabile, situata in 23641 e 23642, contiene l'indirizzo iniziale di qualsiasi testo immesso nel sistema e può essere adoperata per calcolare la quantità di memoria occupata dalle variabili in uso. Supponendo di aver già valutato vars:

```
LET eline = PEEK 23641 + 256 * PEEK23642
PRINT eline—vars
```

visualizza lo spazio occupato dalle variabli.

- chans: questa variabile fornisce l'indirizzo iniziale delle informazioni sui canali ed è situata nelle locazioni 23631 e 23632. L'uso di questa variabile è spiegato con maggiori dettagli più avanti.

- stkend: il valore di questa variabile, contenuta in 23653 e 23654, usato assieme a quello di RAMTOP, consente di calcolare la memoria ancora libera per i programmi e le variabili BASIC.

La formula è:

```
LET stkend = PEEK23653 + 256 * PEEK23654
```

Avendo calcolato RAMTOP, la memoria libera risulta da:

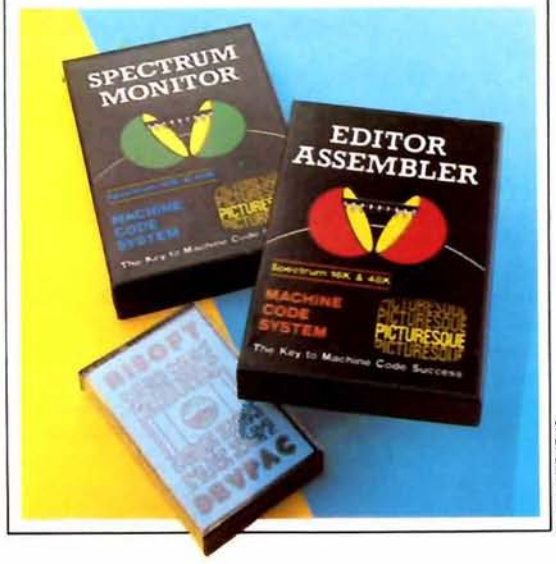
```
PRINT ramtop—stkend
```

Alternativamente, si può usare una routine ROM (anche se il risultato fornito dal metodo appena descritto è più preciso), basta impostare, dopo aver estratto il valore di RAMTOP:

```
PRINT ramtop—USR 7962.
```

PROGRAMMI ASSEMBLER

Per lo Spectrum della Sinclair esistono in commercio svariati package per la produzione di programmi in codice macchina. Uno dei più diffusi è il "DEVPAC 3", realizzato dalla Hisoft.
Fornito su cassetta o in versione per Microdrive, esso comprende l'assembler GENS3 e il monitor/debugger MON3. Il "DEVPAC 3" è molto economico, ma il suo uso non è particolarmente "amichevole".
Sempre per lo Spectrum la Picturesque offre un Editor, un Assembler e un Monitor (inclusivo di disassemblatore) in due package separati.
Un grosso vantaggio di questo sistema è che lo si può usare anche su apparecchi da 16 Kbyte. I manuali di accompagnamento e i programmi hanno un aspetto più professionale rispetto al "DEVPAC 3", sebbene vi sia poca differenza di prezzo



CHRIS STEVENS

ASTA DI MISURAZIONE

Una carenza dello Spectrum è la mancanza di un comando che fornisca la lunghezza di un programma in memoria. Questa informazione può essere estratta tramite opportune PEEK, ma la routine presentata qui sotto fa molto di più: calcola la lunghezza del programma in memoria e poi la deposita nella linea 1 REM dello stesso programma. Unico requisito è che i primi caratteri della prima linea del programma sotto verifica siano 1 REM 00000, come ad esempio in:

1 REM 00000 = lunghezza del programma

Dopo l'esecuzione della routine, tale linea contiene il numero di byte usati dal programma, ad esempio:

1 REM 00801 = lunghezza del programma

Dopodiché basta operare una normale SAVE del programma per registrare in modo permanente, nel suo listato, il valore della lunghezza. Il primo listato riprodotto è in Assembly, mentre il secondo è in BASIC, per chi non possiede un assembler.
In questa seconda versione, il comando CLEAR 61999 modifica il valore di RAMTOP, cosicché la prima locazione RAM "sicura" è la 62000. È da questo indirizzo che inizia la routine in codice macchina. Il valore di RAMTOP resta inalterato finché non viene impartito un nuovo comando CLEAR n, oppure PRINT USR 0 o, infine, l'apparecchio viene spento e poi riacceso, mentre il comando NEW non interferisce con RAMTOP. Finché il valore di RAMTOP non viene modificato, la routine in codice macchina è protetta da eventuali sovrascritture da parte del BASIC.
Per utilizzare la routine, occorre impartire un comando USR, usando come argomento l'indirizzo della routine in codice macchina. Poiché questa è stata caricata dalla locazione 62000 in poi, esistono tre formati diversi coi quali innescare la routine:

RANDOMISE USR(62000)
PRINT USR(62000)
LET L=USR(62000)

Detto per inciso, usando i formati con PRINT e LET al BASIC viene restituito un valore, corrispondente al contenuto del registro BC del processore. In questa routine tale valore non interessa, ma in altre applicazioni può tornare utile

Listato Assembly

```
62000                10
62000  2A4B5C        20              LD    HL,(23627)  ;deposita VARS in HL
62003  ED5B535C      30              LD    DE,(23635)  ;deposita PROG in DE
62007  AF            40              XOR   A           ;azzera flag di carry
62008  ED52          50              SBC   HL,DE       ;lung. programma in HL
62010  DD2A535C      60              LD    IX,(23635)  ;inizio del program in IX
62014  DD23          70              INC   IX
62016  DD23          80              INC   IX          ;incrementa IX per puntare
62018  DD23          90              INC   IX          ;al carattere dopo il token
62020  DD23          100             INC   IX          ;della REM nella prima linea
62022  DD23          110             INC   IX
62024  011027        120             LD    BC,10000    ;estrae i codici ASCII di
62027  CD6BF2        130             CALL  PRINT       ;ciascuna cifra nella lung.
62030  DD23          140             INC   IX          ;del programma e li deposita
62032  01E803        150             LD    BC,1000     ;nelle REM
62035  CD6BF2        160             CALL  PRINT
62038  DD23          170             INC   IX
62040  016400        180             LD    BC,100
62043  CD6BF2        190             CALL  PRINT
62046  DD23          200             INC   IX
62048  010A00        210             LD    BC,10
62051  CD6BF2        220             CALL  PRINT
62054  DD23          230             INC   IX
62056  010100        240             LD    BC,1
62059  AF            250  PRINT:     XOR   A           ;azzera carry e contatore
62060  B7            260  L1:        OR    A
62061  ED42          270             SBC   HL,BC       ;sottrae potenza di dieci dalla
                                                       ;lunghezza del programma..
62063  3803          280             JR    C,FINISH    ;e salta se risult. negativo
62065  3C            290             INC   A           ;incr. reg. A se positivo
62066  18F8          300             JR    L1          ;ripete
62068  09            310  FINISH:    ADD   HL,BC       ;riporta la lung. a un
                                                       ;valore positivo per la
                                                       ;successiva potenza di 10
62069  C630          320             ADD   A,48        ;estrae codice ASCII della cifra
62071  DD7700        330             LD    (IX),A      ;deposita ASCII nella REM
62074  C9            340             RET               ;lavoro concluso
```

Caricatore BASIC

```
  10 REM 00000 = Lunghezza in Byte
  20 CLEAR 61999
  30 FOR i=0 TO 74
  40 READ a: POKE 62000+i,a
  50 NEXT i
1000 DATA 42,75,92,237,91,83,92,175,237,82,221,42,83,
     92,221,35,221,35,221,35,221,35,1,16,39,205,
     107,242,221,35,1,232,3,205,107,242,221,35,1,100,0
1010 DATA 205,107,242,221,35,1,10,0,205,107,242,221,35,
     1,1,00,175,183,237,66,56,3,60,24,-8,9,198,48,221,
     119,0,201
```

Nota: Per garantire un corretto posizionamento delle cifre nella prima linea REM del programma, non si inseriscano spazi tra il numero di linea e la parola chiave REM, oltre a quello automaticamente inserito dall'editor dello Spectrum

"Minder"

Un gioco di contrattazione commerciale in cui dimostrare la propria abilità

Tutti, maledetti e subito

Le figure mostrano i quattro ambienti visitati durante il gioco, il cui scopo è quello di guadagnare il massimo, comprando e rivendendo merce "poco pulita" in locali pubblici malfamati. Se si esauriscono le merci tenute in magazzino, ci si può rivolgere ad appositi fornitori, sempre contrattando i prezzi fino all'ultimo



Il Winchester Bar



La contrattazione



Il magazzino



Inventario delle merci

LIZ HEANEY

Il gioco *Minder* è basato sulle torbide attività di un personaggio chiamato Arthur Daley e del suo aiutante e seguace Terry McCann (entrambi protagonisti di una seguitissima serie televisiva britannica). Il giocatore assume il ruolo di Arthur e lo scopo del gioco è di guadagnare più denaro possibile durante quattordici giorni di accanite contrattazioni di compra/vendita.

Le merci da acquistare e da rivendere sono le più disparate (scatolette di cibo per animali, gingilli "nucleonici", e via dicendo) e i potenziali clienti cercano sempre di tirare sul prezzo.

Le trattative vengono svolte in locali pubblici di dubbia fama nei quali, per complicare le cose, può farsi vivo il Sergente di Polizia Chisholm.

Può infatti capitare che nel bel mezzo di una contrattazione per una partita di home computer, ad esempio, il sergente Chisholm entri nel locale spargendo la voce di un grosso furto... guarda caso proprio di home computer! Ovviamente, il valore delle offerte di acquisto precipita notevolmente.

La faccenda si complica ancora di più se quel bravo sergente di Polizia scova il magazzino di Arthur e vi trova merce di provenienza furtiva.

Le contrattazioni per gli acquisti e le prese di contatto con Terry, che si occupa del trasporto della merce da un luogo all'altro, avvengono nel Winchester Bar. In questo luogo, come in ogni altro del gioco, ci sono fino a sei persone con le quali avviare una conversazione.

Le loro facce compaiono sullo schermo e per selezionarne una basta premere il tasto ad essa corrispondente.

Quando si desidera "parlare" con Terry o con Dave (il barman) tale scelta è superflua poiché sono gli stessi personaggi ad offrire le merci più impensabili (quali, ad esempio, conigliere in mogano o impermeabili di plastica "fondente").

La strategia da adottare per gli acquisti è simile a quella per le vendite: prima ci si informa sul prezzo e sulla quantità delle merci poi si contratta.

Durante le trattative un orologio scandisce il tempo (il potenziale venditore o acquirente se ne va via se non conclude l'affare entro un certo tempo).

Se è il giocatore a voler interrompere il dialogo, basta digitare BYE e la persona si allontana.

Terry è un tipo spesso inafferabile e, quando porta a termine un incarico, si aspetta di ricevere una gentilezza (un bicchierino o un premio) e nel contrattare i prezzi delle merci va tenuto conto anche di ciò.

Senza dubbio la parte più divertente del gioco (posto che si conosca la lingua inglese) è proprio la contrattazione, poiché sullo schermo compaiono gustosi commenti della controparte, destinati, a seconda del caso, ad aumentare o abbassare il valore della merce in questione.

Quando si offre un prezzo, è necessario rispettare la ferrea sintassi imposta dal gioco, altrimenti il computer si rifiuta di capire.

Tutto considerato questo gioco è piuttosto originale e si discosta un po' dai soliti "smanettamenti" di joystick o dalle avventure epiche.

Minder: Per Spectrum Sinclair, MSX, Memotech, Amstrad e Commodore 64
Editori: DK'tronics (GB)
Autore: Don Priestley
Formato: Cassetta
Joystick: Non richiesti

Esperto a portata di chip

L'Intelligenza Artificiale: ingrediente essenziale di un sistema esperto

Il tumultuoso sviluppo della scienza e della tecnica implica la necessità di esperti sempre più specializzati, la cui preparazione richiede lunghi anni di studio e di tirocinio. Per di più, i professionisti veramente preparati sono rari, ricevono alti stipendi e non sempre riescono a rendere partecipi gli altri della loro esperienza, che rischia di andar perduta alla loro morte. Per tutti questi motivi alcuni studiosi di informatica hanno pensato di sviluppare programmi capaci di incamerare tramite computer questo immenso patrimonio di esperienze e conoscenze.

L'idea di creare "sistemi esperti" nacque intorno agli anni '70, quando gli studiosi di Intelligenza Artificiale abbandonarono o rinviarono ad un futuro più promettente la progettazione di macchine genericamente "intelligenti" e decisero di impiegare le teorie sviluppate per risolvere problemi di maggior concretezza e attualità. Quindi i sistemi esperti sono uno dei primi esempi di AI applicata e le tecniche su cui si basano hanno aperto nuovi orizzonti ai laboratori di ricerca, inoltre hanno permesso di finalizzare l'Intelligenza Artificiale a molte applicazioni di interesse pratico. Oggi esistono sistemi esperti capaci di sostituire l'uomo nella diagnosi di malattie, nell'interpretazione di spettrogrammi di massa e in molti altri campi.



MARCUS WILSON-SMITH

Cos'è un "sistema esperto"?

Di solito un sistema esperto si basa su un vasto archivio di conoscenze, relative ad un argomento specifico. In genere, le conoscenze sono organizzate in forma di principi informativi, che consentono al sistema di trarre conclusioni in base a una serie d'informazioni o premesse e di formulare giudizi "ragionevoli" o prendere decisioni. La progettazione di tali sistemi basati sulla conoscenza è una vera evoluzione nella scienza dei computer e ne preannuncia sviluppi rivoluzionari. Questo nuovo approccio sostituisce alla tradizionale formula: "Informazione + algoritmo = Programma" una nuova architettura incentrata su una "banca di conoscenza" e su un "motore d'inferenza", articolata nella formula: "Conoscenza + Inferenza = Sistema Esperto", apparentemente simile alla precedente ma assai diversa nei risultati pratici. Per meglio capire cosa sia un sistema esperto, ecco una tabella sintetica che ne delinea le caratteristiche salienti:

- Un sistema esperto si articola mediante un ventaglio di esperienze relativamente limitato.
- Deve saper "ragionare" su dati incerti e regole non sempre affidabili.
- Deve essere capace di spiegare in modo chiaro la successione dei ragionamenti.
- I "fatti" e il meccanismo d'inferenza sono indipendenti: la conoscenza non viene codificata "fisicamente" all'interno di procedure deduttive.
- E' progettato per apprendere dalla propria esperienza.
- Deve funzionare in base a regole.
- Le sue risposte sono comunicazioni scritte, non tabelle di cifre o grafici.

La parola chiave è "conoscenza". Quindi un sistema intelligente, capace di risolvere problemi, deve escludere a priori la ricerca cieca o casuale. Per conseguire questo obiettivo, il sistema deve avvalersi degli stessi vantaggi di cui gode un "esperto" rispetto ad un principiante: l'esperienza, ossia conoscenza organizzata. Quindi il sistema deve conoscere i fatti, le regole di inferenza e le diverse strategie di soluzione. Gli elementi essenziali di un buon sistema esperto sono quattro:

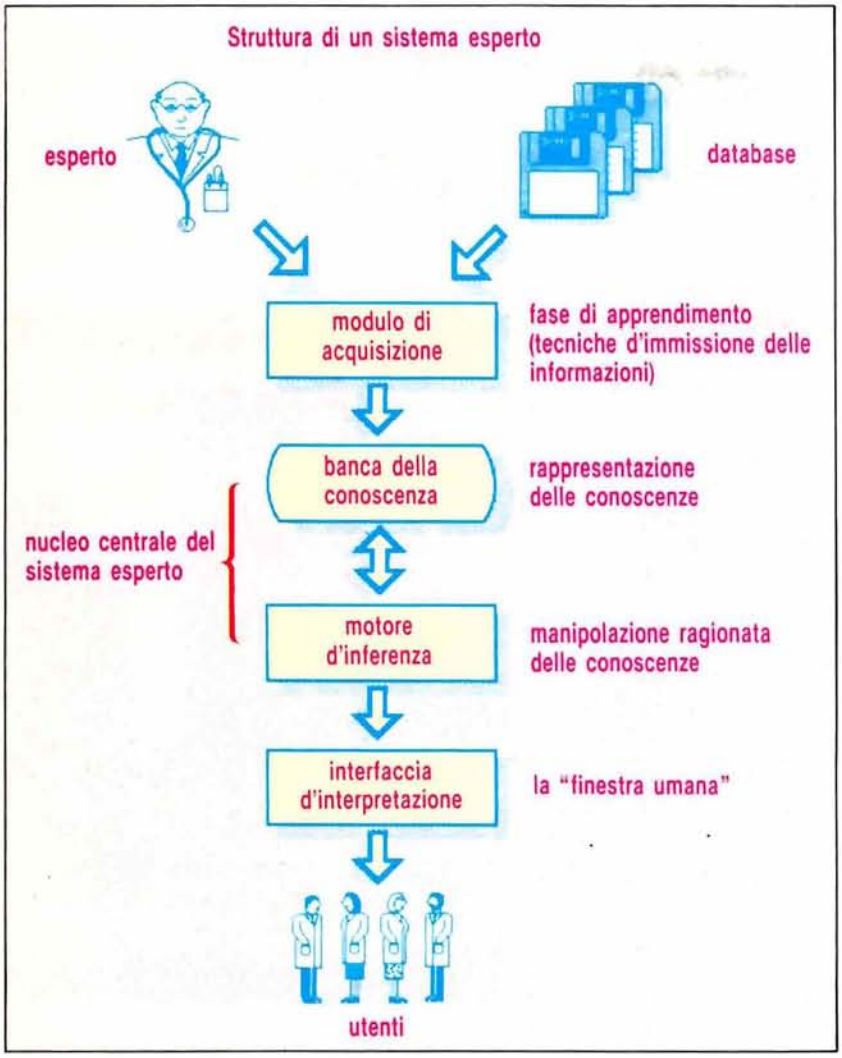
1. La banca di conoscenza
2. Il motore d'inferenza
3. Il modulo d'acquisizione delle conoscenze
4. L'interfaccia d'interpretazione.

Tutti e quattro sono indispensabili e, contrariamente a quanto accade nei sistemi basati

Parola d'esperto

I sistemi capaci di acquisire conoscenze da esperti umani e di usarle per fornire consigli o diagnosi trovano una vasta applicazione in molte discipline: nella medicina, nell'agricoltura, nell'architettura, ecc. I sistemi esperti, progettati per operare all'interno di aree circoscritte, offrono ai professionisti un servizio di consulenza indirizzato specificamente alle loro attività

IAN McKINNELL con il Macintosh



Architettura del sistema

Un sistema esperto si compone di più moduli che consentono il trasferimento delle conoscenze dall'esperto all'utente. Le informazioni fornite dagli esperti vengono registrate in un "database della conoscenza", al quale il sistema applica una serie di procedure d'inferenza per produrre previsioni, consigli o diagnosi. Infine l'interfaccia d'interpretazione, consente all'utente di comunicare col sistema per consultarlo

sulla conoscenza, un vero sistema esperto deve possederli tutti. Conviene quindi analizzarne il funzionamento individualmente.

La banca di conoscenza

Le due componenti fondamentali di un sistema esperto sono la banca della conoscenza e il motore d'inferenza. La prima contiene le informazioni relative all'argomento trattato, che non costituiscono un insieme passivo di registrazioni e dati come quello di un database convenzionale, ma contengono invece rappresentazioni simboliche delle norme di valutazione e delle conoscenze del sistema, espresse in un formato che permette al motore d'inferenza di trarre deduzioni logiche.
La maggior parte delle informazioni contenute nella banca di conoscenza non è matematica e le due difficoltà principali relative alla sua creazione riguardano la rappresentazione e l'acquisizione delle conoscenze. Il primo problema riguarda il metodo da seguire per codificare le conoscenze in modo da renderle comprensibili al computer. Di solito gli elementi da rappresentare sono i seguenti: i termini della materia trattata (in gergo: il "dominio"); le correlazioni strutturali (le interconnessioni tra le diverse informazioni); le correlazioni causali (cioè i loro rapporti di causa-effetto). Il progettista della banca di conoscenza deve quindi scegliere la tecnica più appropriata per registrare queste informazioni in modo simbolico. Le tecniche maggiormente usate sono quattro:

- Regole espresse in formato IF... THEN. La condizione definisce una certa configurazione e il risultato può essere un'azione o una affermazione.
- Reti semantiche. Le connessioni tra i nodi della rete rappresentano le relazioni tra gli argomenti del dominio (ad esempio, la balena è un mammifero).
- Cornici. Queste sono strutture polivalenti di registrazione, che possono contenere valori valutati a priori o codici relativi ad azioni in particolari campi o intervalli.
- Clausole bivalenti (dette anche "a corna"). E' una particolare forma di logica predicata, alla base del linguaggio PROLOG che le usa per eseguire le inferenze (pag. 1752).

I primi sistemi esperti usavano quasi esclusivamente la prima tecnica. Un tipico esempio della struttura IF... THEN è la semplice norma procedurale adottata dal sistema Mycin per diagnosticare le infezioni del sangue:

IF (SE):
1. L'infezione da curare è una meningite, e (AND)
2. L'infezione è di tipo parassitario, e (AND)
3. Gli organismi non sono stati evidenziati dal colorante della coltura, e (AND)
4. Il paziente è un portatore sano, e (AND)
5. Il paziente ha visitato una regione in cui le coccidiomicosi sono endemiche, e (AND)
6. Il paziente è di razza nera o asiatica o indiana, e (AND)
7. L'antigene criptococcico risulta assente dal liquido cerebrospinale

THEN (ALLORA):
Ci sono ragionevoli condizioni per dimostrare che il criptococco non è uno degli agenti eziologici che potrebbero aver causato l'infezione.

Questo esempio evidenzia come un sistema esperto usi il gergo tecnico tipico del campo nel quale è destinato a operare, in questo caso la medicina. La struttura IF... THEN usata dal Mycin è composta essenzialmente da una serie di istruzioni che possono dimostrarsi vere o false. Quindi tali istruzioni possono essere collegate da operatori Booleani (quali AND) che aiutano l'elaborazione del computer. Per selezionare le informazioni necessarie a una corretta diagnosi, il sistema deve poter dialogare con l'operatore umano. Nel caso del Mycin, l'utente deve possedere conoscenze mediche per capire le domande del sistema esperto e rispondere adeguatamente.

Il motore d'inferenza

Il meccanismo di inferenza si basa sulla ricerca e su tecniche di ragionamento che consentono al sistema di trovare una soluzione e, se necessario, di motivare le proprie risposte. Esistono due strategie globali di ragionamento, dette concatenazione "deduttiva" e "induttiva".

La concatenazione induttiva parte dai dati (sintomi) e giunge a una conclusione (diagnosi). In un sistema basato su norme, questa tecnica confronta le condizioni IF con i dati seguendo un ordine stabilito. La concatenazione induttiva può essere facilmente computerizzata e si adatta alla raccolta e all'elaborazione di sequenze di dati, come per i valori generati automaticamente da uno strumento o nella compilazione di moduli.
La concatenazione deduttiva parte invece da ipotesi per giungere alla dimostrazione di un fatto. Il sistema formula un'ipotesi e valuta i dati che la avvalorano o la negano. Può essere programmato in modo ricursivo e nei sistemi a consultazione porta a un tipo di dialogo più naturale. Poiché non si è ancora risolto completamente il problema di scegliere l'ipotesi iniziale più opportuna nelle diverse situazioni, la maggior parte dei sistemi esperti ricorre a una combinazione delle concatenazioni deduttiva e induttiva.

Il modulo d'acquisizione

Gli esperti umani sono generalmente restii a comunicare le proprie conoscenze, non tanto perché vogliano mantenerle segrete, quanto perché i loro processi di pensiero si sviluppano molto spesso in modo inconscio e intuitivo. Quindi l'acquisizione delle conoscenze è stato uno dei problemi di più difficile soluzione nello sviluppo dei sistemi esperti. Di recente si è prestata particolare attenzione allo sviluppo di software che permetta ad un sistema esperto di costruirsi un proprio criterio di conoscenza, sulla base di esempi preordinati. L'impiego di questa tecnica ha facilitato il processo di acquisizione delle conoscenze eliminando contemporaneamente molte delle difficoltà incontrate nel trasferimento ai sistemi delle conoscenze umane e il lungo lavoro di codifica per il computer. Sebbene i sistemi esperti non siano ancora capaci di svolgere l'intero processo di acquisizione, passando da un database di esempi a una serie di regole decisionali, sono tuttavia in grado di affinare la loro base di conoscenza durante un periodo di "apprendistato" o durante il lavoro, imparando dai propri errori.
Alcuni dei sistemi di apprendimento del computer vengono esaminati meglio in seguito.

L'interfaccia d'interpretazione

Una caratteristica del Mycin, del Prospector e di altri sistemi esperti è la capacità di giustificare, a richiesta dell'utente, le proprie conclusioni. Quando la responsabilità di una decisione importante viene lasciata al computer, il sistema deve motivarla adeguatamente. Nell'esempio è riportato un dialogo tra Mycin e utente (che essendo un medico, non accetterà una diagnosi senza le debite spiegazioni). Il dialogo inizia con una domanda del Mycin, alla quale segue la richiesta di giustificazione da parte del medico.

IL "WHEAT COUNSELLOR"



Il "Wheat Counsellor" viene offerto gratuitamente dal sistema Prestel...



Il menu iniziale permette di scegliere tra interventi pianificati di mezza stagione e autunnali...



Il sistema rivolge all'agricoltore una serie di domande e elabora i propri suggerimenti basandosi sulle risposte...



Interpretate le risposte dell'agricoltore, il Wheat Counsellor visualizza le modalità d'impiego dei vari trattamenti chimici disponibili

Per concessione de ICI PLANT PROTECTION UK DIVISION

Il paziente ha contratto la meningite in ospedale?
PERCHÉ?
[2.0] Questa informazione contribuisce a individuare l'organismo (diverso da quelli osservati in coltura o al microscopio) che potrebbe essere il vettore dell'infezione. È già appurato che:
[2.1] l'infezione da curare è una meningite, e (AND)
[2.2] gli organismi non sono stati rivelati dal colorante della coltura in sospensione del liquido cerebro-spinale, e (AND)
[2.3] l'infezione è di tipo batterico, e (AND)
[2.4] l'anamnesi del paziente non evidenzia interventi di tipo neurochirurgico, e (AND)
[2.5] il paziente ha un'età superiore a 10 giorni.
Pertanto, se (IF):
[2.6] il paziente ha contratto l'infezione in ospedale,
allora abbiamo le prove che gli organismi (diversi da quelli osservati in coltura o al microscopio) vettori dell'infezione potrebbero essere: Escherichia.Coli (0.75), Staphilococcus-coag-pos (0.3), Pseudomonas-aeruginosa (0.3), Klebsiella-pneumoniae (0.5)
— da [REGOLA 545]

Questo lungo messaggio è in realtà una parziale esposizione dei processi logici del programma, redatta in gergo medico. I sistemi basati su regole impiegano pochi secondi per visualizzare tali spiegazioni (tratte dalle regole adottate): ciò rappresenta un punto a favore della tecnica di codifica della conoscenza basata su regole. Si noti che le conclusioni tratte nell'esempio precedente sono associate a valori numerici: questi non sono vere percentuali di probabilità ma valutazioni che consentono al sistema di manipolare in modo coerente dati incerti e presentare nell'analisi finale una tabella ordinata di probabili diagnosi.

Un aiuto all'agricoltura

Il "Wheat Counsellor" è un sistema esperto sviluppato dalla ICI e offerto gratuitamente dal Prestel. Il sistema informa gli agricoltori sulle tecniche per combattere le malattie del raccolto, sulle analisi chimiche più adeguate ed esegue una stima delle probabili perdite. Benché il sistema usi un linguaggio confidenziale, il suo database di conoscenza poggia su studi scientifici sulla diffusione delle malattie del frumento. Il Wheat Counsellor rivolge all'utente una serie di semplici domande e trae le proprie conclusioni sulla base delle risposte ottenute. Le sue comunicazioni contengono tabelle sui trattamenti chimici disponibili e sulle loro modalità d'impiego

I file nel CP/M

L'importanza delle cosiddette "estensioni" ai nomi dei file

Come accennato in precedenza, il nome di un file CP/M si compone di una prima parte (il nome vero e proprio, o "primario", per un massimo di 8 caratteri), seguito da un punto e da non più di tre lettere che costituiscono l'"estensione" del file (il cui impiego è facoltativo). Teoricamente a un nome primario può essere aggiunta qualsiasi estensione ma, in pratica, alcune di esse sono fissate per convenzione. Ad esempio l'estensione .COM, riservata ai file di comando del CP/M, comunica al sistema operativo che questo file deve essere aggiunto alla lista dei comandi transitori eseguibili sotto CP/M. Nel gergo dei programmatori si parla di file "punto COM" o "punto BAS", anche se il punto non è propriamente parte dell'estensione.
Analogamente, i programmi BASIC scritti sotto CP/M vanno archiviati con l'estensione .BAS, poiché è questa l'estensione che la maggior parte delle versioni di BASIC richiedono. In genere l'interprete BASIC consente di memorizzare su disco i listati dei programmi secondo due formati: ASCII (normale file di testo) oppure testo frammisto a "token" (i codici numerici corrispondenti alle parole chiave del BASIC). Se invece si adoperano compilatori BASIC, il file "sorgente" è sempre nel formato ASCII, ma quello prodotto dalla compilazione ha l'estensione .INT, che indica un file in codice "intermedio", ossia non immediatamente eseguibile, se non tramite un apposito modulo di esecuzione.
Anche per i programmi in codice macchina vengono usate estensioni diverse, a seconda del loro stato. Ai listati sorgenti di programmi Assembly, ad esempio, è riservata l'estensione .ASM o .MAC (a seconda dell'assemblatore usato). Se il listato non possiede la corretta estensione, l'assemblatore non trova il file e visualizza un messaggio di errore.
L'assembler, a sua volta, può produrre file di diverso genere, ognuno dei quali possiede lo stesso nome primario del file sorgente, ma un'estensione diversa. I file .HEX, ad esempio, contengono il codice esadecimale, espresso in ASCII, corrispondente al sorgente. I file .PRN contengono un listato identico al sorgente, ma corredato del codice esadecimale corrispondente assieme a una "tabella dei simboli" (ossia le etichette adoperate e relativi indirizzi o valori) e un elenco degli errori riscontrati in fase di assemblaggio. I file .PRN, quindi, sono molto utili in fase di debugging dei programmi assembly.
Esiste infine un gruppo di estensioni associate ai file di testo. In tutte le versioni di CP/M c'è un elementare programma per la redazione di testi (di solito è il comando ED) e in commercio esiste una vastissima scelta di word processor, adatti al CP/M, che offrono sofisticate funzioni per l'elaborazione di testi. Tuttavia, anche tra tutti questi programmi c'è una convenzione per le estensioni dei file. Escludendo quelle già ricordate, l'estensione di un file di testo può essere composta da qualsiasi combinazione di tre lettere, oppure mancare del tutto; in questo caso, alcuni programmi ne assegnano una automaticamente. L'utility TEXTED, ad esempio, assegna al nome primario di un file l'estensione .TXT.

I file "storici" o di "backup"

Se si esamina la directory del dischetto, subito dopo aver elaborato un file di testo, si nota che esistono almeno due file con lo stesso nome: uno con l'estensione scelta dall'operatore, l'altro con l'estensione di tipo .BAK. I text editor e i word processor, infatti, hanno la buona abitudine di non cancellare la versione precedente l'ultimo intervento apportato: essa viene copiata, appunto, nel file .BAK. Se il file originale "di lavoro" viene accidentalmente danneggiato, ne esiste sempre una versione di riserva, anche se non aggiornata con l'ultimo intervento. È buona norma, comunque, provvedere alla copia del testo originale anche su di un altro dischetto, per avere sempre una copia di riserva.
Pur non essendo obbligatorie, le estensioni facilitano l'organizzazione dei file perché forniscono informazioni sul loro contenuto e consentono l'impiego fondamentale di particolari caratteri di comando, detti "wildcard" o "jolly" che permettono di intervenire contemporaneamente su più file.
Supponiamo, ad esempio, di avere registrato

CARATTERI DI CONTROLLO DEL CP/M

Il CP/M prevede l'uso di caratteri di controllo per l'esecuzione di molte sue funzioni. Sebbene alcuni di essi appaiano obsoleti e vengano usati di rado, non sono stati eliminati dal sistema standard per mantenere un alto grado di compatibilità tra computer differenti

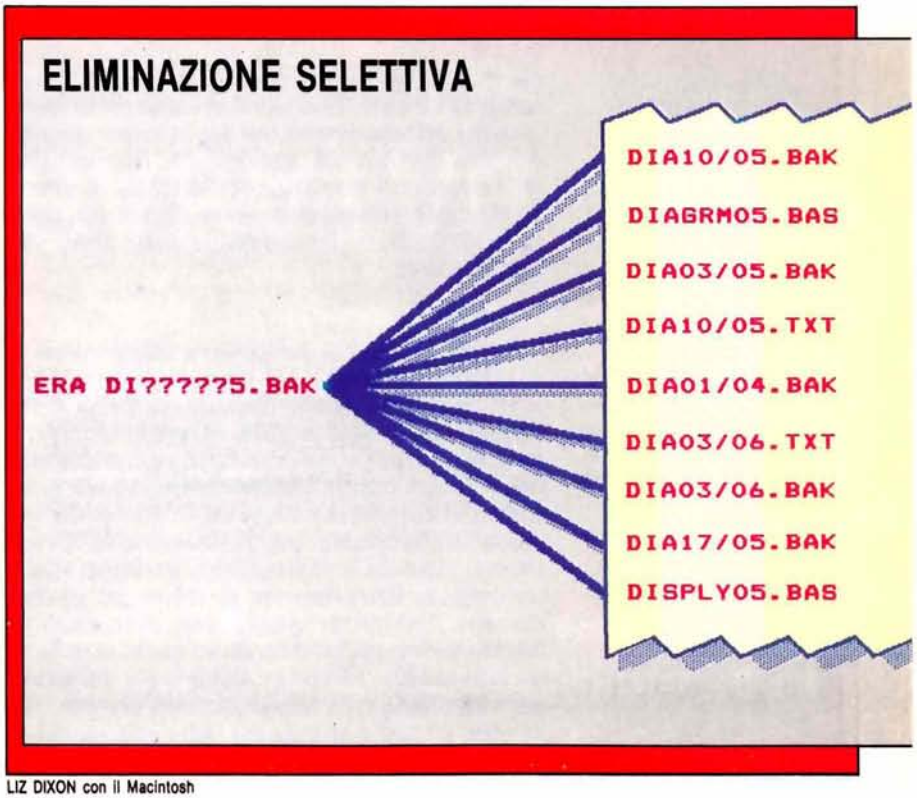
CTRL-C	Interrompe il programma in esecuzione e ricarica il sistema dal dischetto CP/M
CTRL-C	Inizializza il disco che viene inserito nel drive
CTRL-M	Esegue la corrente linea di comando (usato al posto di RETURN)
CTRL-M	Pone fine ai comandi PIP e restituisce il controllo al CP/M
CTRL-Z	Nel programma ED, serve per tornare a livello comandi dopo un inserimento
CTRL-P	Invia alla stampante il testo sullo schermo. La sua ripetizione disabilita la funzione
CTRL-U	Cancella la linea corrente
CTRL-X	Cancella la linea corrente e sposta il cursore al suo inizio
CTRL-H	Spaziatura all'indietro/cancellazione
CTRL-E	Permette l'immissione di una lunga linea di comando, prima di eseguirla
CTRL-R	Ripete la linea di comando corrente

una serie di documenti relativi al mese di Marzo. Se riteniamo di non averne più bisogno, possiamo scorrere la directory e cancellare individualmente i file che li contengono con il comando ERA. Se però i file sono molti, tale procedimento è lungo e noioso.
Al contrario, se a tutti i file relativi al mese di Marzo è stata assegnata una stessa estensione, ad esempio .MAR, la loro eliminazione è possibile usando un unico comando: ERA D:*.MAR (D rappresenta il nome del drive che contiene il dischetto con i file). L'asterisco prima del punto ordina di cancellare tutti i file con estensione .MAR, qualunque sia il nome primario. L'asterisco può essere usato anche alla destra del punto in sostituzione dell'estensione: ERA LETT5.* cancella tutti i file di nome LETT5, con qualunque estensione.

Cancellazione selettiva dei file

Supponiamo adesso che tutti i file di Marzo possiedano l'estensione .MAR ma che alcuni contengano documenti importanti che non devono essere cancellati. In questo caso l'impiego del simbolo * eliminerebbe anche i file che si vogliono conservare. Tuttavia, se si è avuta l'accortezza di assegnare loro nomi appropriati, quali DOC1.MAR, DOC2.MAR, ecc., è possibile cancellare, tramite un unico comando, solo alcuni file, conservando quelli con i documenti importanti: ERA D:DOC?.MAR ordina al sistema operativo di cancellare tutti i file il cui nome primario inizia con DOC e termina con l'estensione .MAR, qualunque sia il quarto carattere corrispondente al simbolo ? nella quarta posizione.
Il punto interrogativo può comparire in una qualunque delle 11 posizioni possibili del nome del file. Quindi, se vogliamo cancellare sia i file di Marzo (estensione .MAR) che di Maggio (estensione .MAG), possiamo usare il comando ERA DOC?.MA? nel quale il simbolo ? sostituisce l'ultima lettera dell'estensione. I simboli * e ? possono apparire contemporaneamente in un comando. Il comando ERA D:????Q???.* cancella tutti i file nel cui nome primario compare una Q in quinta posizione.
I caratteri "jolly" sono utilizzabili anche nei comandi PIP, STAT e REN. Per trasferire tutti i file da un dischetto a un altro, si può usare il comando PIP B:=A*.*. Questo individua i file del dischetto in A che corrispondono al formato selezionato (in questo caso tutti) e li copia sul dischetto nel drive B.
Fin qui si sono esaminati i comandi interni e quelli transitori del CP/M, ma esistono altri comandi, o per meglio dire speciali caratteri di controllo, utili in svariate circostanze. Forse il carattere di controllo più utile e maggiormente usato è CTRL-C che ripristina in memoria la prima traccia del sistema, ricaricando cioè il CP/M. Tale comando viene impartito non solo per ricaricare il CP/M dopo una caduta del sistema, ma anche quando viene inserito un nuovo dischetto in uno dei drive. Si ricordi che il CP/M conserva nella memoria del computer una copia dell'identificatore del drive corrente. Quando il dischetto inserito in un drive viene sostituito, il sistema operativo deve essere informato del cambiamento, altrimenti può generare degli errori. Quando viene ripristinato il CP/M, esso ricarica in memoria l'identificatore di ogni unità a dischetti, consentendo così un corretto funzionamento del sistema operativo. La combinazione CTRL-C viene anche usata per terminare l'esecuzione di molti programmi e tornare al "prompt" del CP/M.
Molti degli altri caratteri di controllo vengono usati nelle operazioni di editing e sono stati adottati da numerosi word processor che girano sotto CP/M. Ad esempio, CTRL-H elimina l'ultimo carattere battuto, CTRL-U e CTRL-X cancellano un'intera linea di comando e CTRL-P abilita o disabilita la copia su stampante di quanto appare sullo schermo.
Benché l'evoluzione dell'hardware e del software abbia reso obsoleti alcuni dei caratteri di controllo usati in CP/M, vengono conservati anche se esistono tasti che duplicano le loro funzioni. Per esempio, la maggior parte dei computer possiede oggi il tasto Delete e quelli per il posizionamento del cursore. Un motivo che ha indotto a non modificare il sistema è la compatibilità. Mentre il tasto CTRL e i caratteri alfabetici hanno mantenuto nel tempo i loro equivalenti ASCII, qualunque sia la tastiera usata, ciò non è necessariamente vero per i tasti speciali o di funzione aggiunti in seguito: i caratteri di controllo conservano quindi un alto grado di compatibilità. Infine, gli utenti abituati a lavorare in CP/M incontrerebbero grosse difficoltà se il sistema venisse modificato.
In seguito viene esaminata la struttura interna del CP/M e l'interazione tra sistema operativo, computer e periferiche collegate.

Uso dei caratteri "wildcard"
Introducendo questi caratteri nelle diverse posizioni dell'argomento di un comando che esegue una ricerca sulla directory, il computer ignora i caratteri in quella posizione. Quindi, un'oculata assegnazione dei nomi di file permette interventi (di cancellazione o di copia, ad esempio) ricorrendo a un unico comando



LIZ DIXON con il Macintosh

Tracciamenti fantasiosi

La sezione conclusiva del software di controllo per il tracciatore digitale

La seconda parte del programma per il Digital Tracer controlla il suo funzionamento nel modo "a elastico". Questa funzione esclude la possibilità di tracciare figure a mano libera, consentendo invece di specificare i punti iniziale e finale di una linea, il centro e il raggio di una circonferenza oppure i tre angoli di un triangolo internamente colorato. Per selezionare tali funzioni, con quella per il tracciamento di un singolo punto, si preme il tasto corrispondente in un menu visualizzato nella parte superiore dello schermo. Le funzioni di SAVE, LOAD e CLEAR dello schermo (simili a quelle già ricordate per il modo di tracciamento a mano libera) vengono selezionate tramite il medesimo menu. Questa parte del programma consente di visualizzare figure complicate, pilotando il tracer con la tastiera, salvare le immagini sul dischetto e richiamarle in un secondo tempo.

Le quattro procedure

PROCpoint

Per muovere il cursore sullo schermo, ma senza tracciare linee, occorre assegnare il valore 1 alla variabile togflag, che abilita il modo "penna-sollevata", e successivamente chiamare PROCdraw (una procedura sviluppata nella prima parte del programma). Questa chiamata è inserita in un ciclo nel quale viene controllata la pressione di tasto per l'uscita dal ciclo o di un tasto per la modifica del colore di primo piano. Inoltre, quando l'utente preme il pulsante del tracer, la procedura traccia un punto sulla attuale posizione del cursore usando il colore corrente di primo piano. Per annullare le eventuali ripetizioni di battuta, la procedura si interrompe dopo aver rilevato la pressione di un tasto finché quest'ultimo non viene rilasciato.

PROCline

Per prima cosa la procedura memorizza le coordinate correnti del cursore, usate in seguito per calcolare le coordinate finali della linea tracciata nel modo a elastico. Quindi fa sparire il cursore dallo schermo e traccia una linea fino al punto indicato come nuova posizione del cursore. La linea viene disegnata nel modo OR-Esclusivo, con l'istruzione GCOL3, cosicché, ripetendo l'istruzione, la linea viene cancellata, ripristinando il colore di sfondo iniziale. Inserendo queste due procedure di tracciamento all'interno di un ciclo, si ottiene un movimento "elastico" della linea: un'estremità di questa è fissata al punto d'origine specificato e l'altra al cursore sullo schermo, abbinato al tracer. Quando la linea assume la

COMANDI DEL TRACER DIGITALE

Menu principale

Opzione	Funzione
1	Modo "a mano libera"
2	Modo "a elastico"

Menu per il modo "a mano libera"

Opzione	Funzione
S	Salva l'immagine su nastro o dischetto
L	Legge l'immagine da nastro o dischetto
M	Riporta al menu principale
C	Pulisce lo schermo
Pulsante	Penna-sollevata/penna-abbassata

Menu per il modo "a elastico"

Opzione	Funzione
S,M,L,C	Come per il modo a mano libera
P	Traccia un punto nel colore corrente di primo piano
M	Riporta al menu principale
D	Traccia una linea nel corrente colore di primo piano dalla precedente posizione del cursore a quella corrente. Il tasto M o il pulsante del tracer riportano al menu del modo a elastico
F	Quando viene premuto il pulsante, traccia un triangolo fra tre punti specificati
R	Traccia un cerchio nel colore corrente di primo piano. Il centro corrisponde alla posizione del cursore all'inizio della procedura, mentre il raggio dipende dalla posizione corrente del cursore. Premere M o il pulsante per tracciare il cerchio e tornare al menu del modo a elastico

Colori di primo piano

Opzione	Funzione
1	Seleziona il colore logico 1 (se non specificato: rosso)
2	Seleziona il colore logico 2 (se non specificato: giallo)
3	Seleziona il colore logico 3 (se non specificato: bianco)

NB: I colori logici possono essere cambiati usando il comando VDU 19 per selezionare un nuovo colore tra i 16 disponibili. Per esempio, VDU 19,1,6,0,0,0 sostituisce l'azzurro (colore attuale 6) al colore logico 1. I colori di primo piano possono essere variati in qualsiasi momento

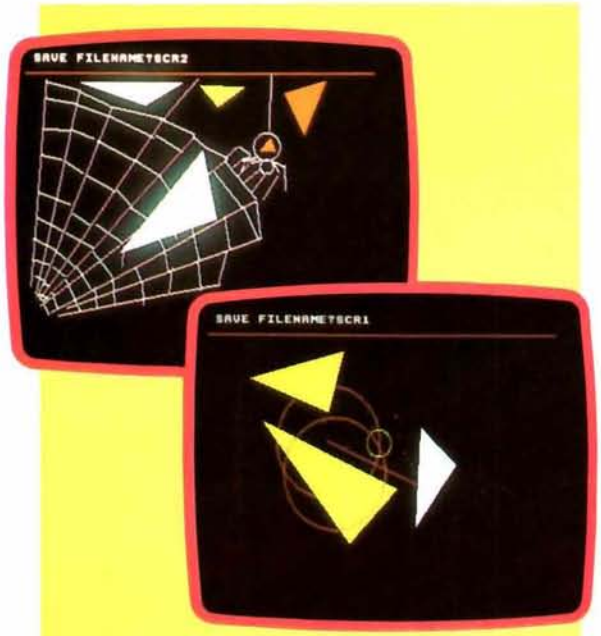
posizione desiderata, l'utente ne fissa l'immagine premendo il pulsante del tracer oppure il tasto M della tastiera. In questo caso la procedura torna al menu principale senza eseguire la seconda istruzione di tracciamento, che cancellerebbe la linea appena "fissata"

PROCfill

La procedura fill si basa sulla funzione di colorazione dei triangoli del Micro BBC, abilitata tramite l'istruzione PLOT 85,x,y. Questa adopera come vertici del triangolo le ultime due posizioni occupate dal cursore e il punto specificato nel comando, riempiendo di colore la figura ottenuta. La procedura segue un metodo simile a quello adottato in PROCpoint e consente di usare il pulsante del tracer per tracciare i tre angoli del triangolo da colorare. Le ultime due linee del listato di questa procedura esaminano ciclicamente le coordinate dei tre punti per eseguire la colorazione del triangolo.

PROCcircle

Questa procedura ricorre al metodo della linea elastica, già usato in PROCline, per consentire la definizione del raggio di una circonferenza. In questo caso, però, PROCline viene richiamata dall'interno della procedura che traccia il cerchio. Specificato il raggio, il controllo torna a PROCcircle, che cancella il cursore e la linea tracciata da PROCline prima di disegnare il cerchio. Le coordinate del centro (x1,y1) sono quelle della posizione del cursore al momento della chiamata alla procedura, mentre PROCline fornisce quelle di un punto sulla circonferenza (x2,y2). PROCcircle, infine, adopera questi due punti per calcolare il raggio e disegna il cerchio con un algoritmo per il tracciamento di una circonferenza.



Realizzazioni grafiche

Dopo aver aggiunto la seconda parte del programma per il tracciatore (listato a fianco) a quella fornita in precedenza, si può usare il sistema per tracciare linee rette, cerchi e triangoli colorati e creare immagini simili a quelle qui riprodotte

PROGRAMMA PER TRACCIATORE DIGITALE 2ª PARTE

Il listato qui a fianco è la seconda parte del programma per il tracciatore digitale per Micro BBC e va aggiunto alla prima parte presentata a pag. 1768

```
2350 DEF PROCmodo_elastico
2360 PROCinforma_elastico
2370 REPEAT
2380 risp$=INKEY$(1): IF risp$<>"" THEN
     PROCpremi_tasto
2390 flagscelta=PROCdraw
2400 UNTIL flaguscita=1
2410 ENDPROC
2430 DEF PROCinforma_elastico
2440 PROCcalcola_xy:vecchiax=x:vecchiay=y:
     PROCcursore(vecchiax,vecchiay)
2450 PROCreinforma
2460 GCOL 0,1: MOVE 0,920: DRAW 1280,920
2470 ENDPROC
2480 DEF PROCreinforma
2490 PRINT TAB(1,1);SPC(79)
2500 PRINT TAB(1,1);"S=Save L=Load M=Menu
     C=Cancella R=Cerchio"
2510 PRINT " D=Traccia Linea P=Punto F=Colora"
2520 ENDPROC
2540 DEF PROCpremi_tasto
2550 IF risp$="C" THEN CLS: PROCinforma_elastico:
     ENDPROC
2560 IF risp$="M" THEN flaguscita=1
2570 PROCcambio_colore
2580 IF risp$="R" THEN PROCtitolo_cerchio:
     PROCcircle: PROCinforma_elastico: ENDPROC
2590 IF risp$="D" THEN PROCtitolo_linea: PROCline
2600 IF risp$="P" THEN PROCtitolo_punto: PROCpoint:
     PROCinforma_elastico: ENDPROC
2620 IF risp$="S" THEN PROCsave_schermo:
     PROCinforma_elastico :ENDPROC
2630 IF risp$="L" THEN PROCload_schermo:
     PROCinforma_elastico :ENDPROC
2640 ENDPROC
2660 DEF PROCtitolo_punto
2670 PRINT TAB(1,1) SPC(79)
2680 PRINT TAB(15,1);"Modo Punto"
2690 ENDPROC
2710 DEF PROCpoint
2720 REPEAT
2730 flagscelta=1:PROCdraw
2740 risp$=INKEY$(1)
2750 PROCcambio_colore
2760 IF (ADVAL(0) AND 3) <>0 THEN PLOT 69,x,y:
     REPEAT UNTIL (ADVAL(0) AND 3)=0
2770 UNTIL risp$="M"
2780 PROCreinforma
2790 ENDPROC
2800 DEF PROCtitolo_linea
2810 PRINT TAB(1,1) SPC(79)
2820 PRINT TAB (15,1);"Modo Linea"
2830 ENDPROC
2850 DEF PROCline
2860 x1=x: y1=y: REPEAT
2880 PROCcalcola_xy:PROCcursore(vecchiax,vecchiay)
2890 x2=x: y2=y: GCOL 3, colore: MOVE x1,y1: DRAW
     x2,y2
2900 risp$=INKEY$(1)
2910 IF (ADVAL(0) AND 3) THEN risp$="M"
2920 PROCcursore(x,y):vecchiax=x:vecchiay=y
2930 IF risp$<>"M" THEN GCOL 3,colore:MOVEx1,y1:
     DRAW x2,y2
2940 PROCcambio_colore
2950 UNTIL risp$="M"
2960 PROCreinforma
2970 ENDPROC
2990 DEF PROCtitolo_colora
3000 PRINT TAB(1,1) SPC(79)
3010 PRINT TAB (15,1);"Modo Colorazione"
3020 ENDPROC
3040 DEF PROCfill
3050 FOR I=1 TO 3
3060 REPEAT
3070 flagscelta=1: PROCdraw: risp$=INKEY$(1)
3075 IF risp$<>"" THEN PROCcambio_colore
2080 UNTIL (ADVAL(0) AND 3) <> 0
3090 PLOT 69,x,y: x(I)=x: y(I)=y
3110 REPEAT UNTIL (ADVAL(0) AND 3)=0:  REM ATTENDI
     IL RILASCIO DEL PULSANTE
3120 NEXT I
3135 PROCcursore(x,y): REM cancella il cursore
3140 FOR I=1 TO 2: PLOT 69, x(I), y(I): NEXT I
3150 PLOT 85,x,y
3160 ENDPROC
3180 DEF PROCtitolo_cerchio
3190 PRINT TAB(1,1) SPC(79)
3200 PRINT TAB (15,1);"Modo Cerchio"
3210 ENDPROC
3230 DEF PROCcircle
3240 PROCline
3250 GCOL 3,colore: MOVE x1,y1: DRAW x2,y2:REM
     CANCELLA UNA LINEA
3260 GCOL 0,colore: REM TORNA AL MODO DI
     TRACCIAMENTO NORMALE
3270 PROCcursore(x2,y2)
3370 raggio=SQR(x2-x1)^2+(y2-y2)^2
3380 MOVE x1+raggio,y1
3390 FOR angolo=0 TO 2* π STEP 0.1
3400 rx=x1+raggio*COS(angolo)
3410 ry=y1+raggio*SIN(angolo)
3420 DRAW rx,ry
3430 NEXT angolo
3435 DRAW x1+raggio,y1
3440 ENDPROC
```

DATABASE

La nona tabella della mappa di memoria del Commodore 64. Per concessione della Commodore Business Machines

INDIRIZZO ESADECIMALE	ETICHETTA	LOCAZIONE DECIMALE	DESCRIZIONE
030C	SAREG	780	**Storage for 6502 .A Register** Deposito per il registro .A del 6502
030D	SXREG	781	**Storage for 6502 .X Register** Deposito per il registro .X del 6502
030E	SYREG	782	**Storage for 6502 .Y Register** Deposito per il registro .Y del 6502
030F	SPREG	783	**Storage for 6502 .SP Register** Deposito per il registro .SP del 6502
0310	USRPOK	784	**USR Function Jump Instr (4C)** Istruz. di salto (4C) per la funzione USR
0311-0312	USRADD	785-786	**USR Address Low Byte/High Byte** Indirizzo (formato lo/hi) della routine USR
0313		787	**Unused** Locazione non usata
0314-0315	CINV	788-789	**Vector: Hardware IRQ Interrupt** Vettore di interrupt hardware IRQ
0316-0317	CBINV	790-791	**Vector: BRK Instr. Interrupt** Vettore: Interrupt a seguito di BRK
0318-0319	NMINV	792-793	**Vector: Non-Maskable Interrupt** Vettore: Interrupt non mascherabile
031A-031B	IOPEN	794-795	**KERNAL OPEN Routine Vector** Vettore per routine OPEN del KERNAL
031C-031D	ICLOSE	796-797	**KERNAL CLOSE Routine Vector** Vettore per routine CLOSE del KERNAL
031E-031F	ICHKIN	798-799	**KERNAL CHKIN Routine Vector** Vettore per routine CHKIN del KERNAL
0320-0321	ICHKOUT	800-801	**KERNAL CHKOUT Routine Vector** Vettore per routine CHKOUT del KERNAL
0322-0323	ICLRCH	802-803	**KERNAL CLRCHN Routine Vector** Vettore per routine CLRCHN del KERNAL
0324-0325	IBASIN	804-805	**KERNAL CHRIN Routine Vector** Vettore per routine CHRIN del KERNAL
0326-0327	IBSOUT	806-807	**KERNAL CHROUT Routine Vector** Vettore per routine CHROUT del KERNAL
0328-0329	ISTOP	808-809	**KERNAL STOP Routine Vector** Vettore per routine STOP del KERNAL
032A-032B	IGETIN	810-811	**KERNAL GETIN Routine Vector** Vettore per routine GETIN del KERNAL

L'Atari 130XE

Anatomia di un microcomputer a diffusione europea: tecnologie a confronto



CHRIS STEVENS

Un nuovo "look"

L'Atari 130XE è il primo apparecchio lanciato dalla compagnia da quando Jack Tramiel ne ha assunto la presidenza. Possiede una RAM interna di 128 Kbyte e viene venduto a un prezzo inferiore a quello di molti altri home computer che offrono soltanto metà della sua capacità di memoria. La Atari punta su questa nuova macchina per rilanciare il proprio nome sul mercato dei microcomputer

La Atari è stata per molto tempo un'azienda di punta nel mercato degli home computer, ma agli inizi degli anni '80 ha incontrato non poche difficoltà.

Dopo aver subito ingenti perdite, nel 1984 la presidenza della Atari è stata assunta da Jack Tramiel, già direttore della Commodore, che ha tentato di risollevare le sorti di questo gigante dell'informatica impostando una politica commerciale basata su una forte riduzione dei prezzi.

Tuttavia questa scelta non ha prodotto i risultati sperati e la Atari si è trovata a dover affrontare un problema che accomuna molti fabbricanti di computer: il calo delle vendite ha causato una riduzione del software in commercio il quale, a sua volta, ha provocato un'ulteriore caduta delle vendite e una conseguente mancanza di capitali da investire in nuovi modelli.

Nonostante queste difficoltà, nel 1985 la Atari ha annunciato il lancio di una nuova serie di apparecchi, il primo dei quali è stato l'Atari 130XE, un computer basato sul processore 6502C.

In sostanza, l'Atari 130XE è un modello perfezionato dei precedenti apparecchi Atari a 8 bit, lanciati sul mercato in diverse versioni sin dall'inizio del 1980; tuttavia si distingue dai suoi predecessori per il nuovo profilo e per le notevoli dimensioni della memoria interna: possiede infatti una RAM da 128 Kbyte!

L'involucro dell'apparecchio, che è stato completamente trasformato, ha un elegante design ed è realizzato in materiale plastico grigio chiaro.

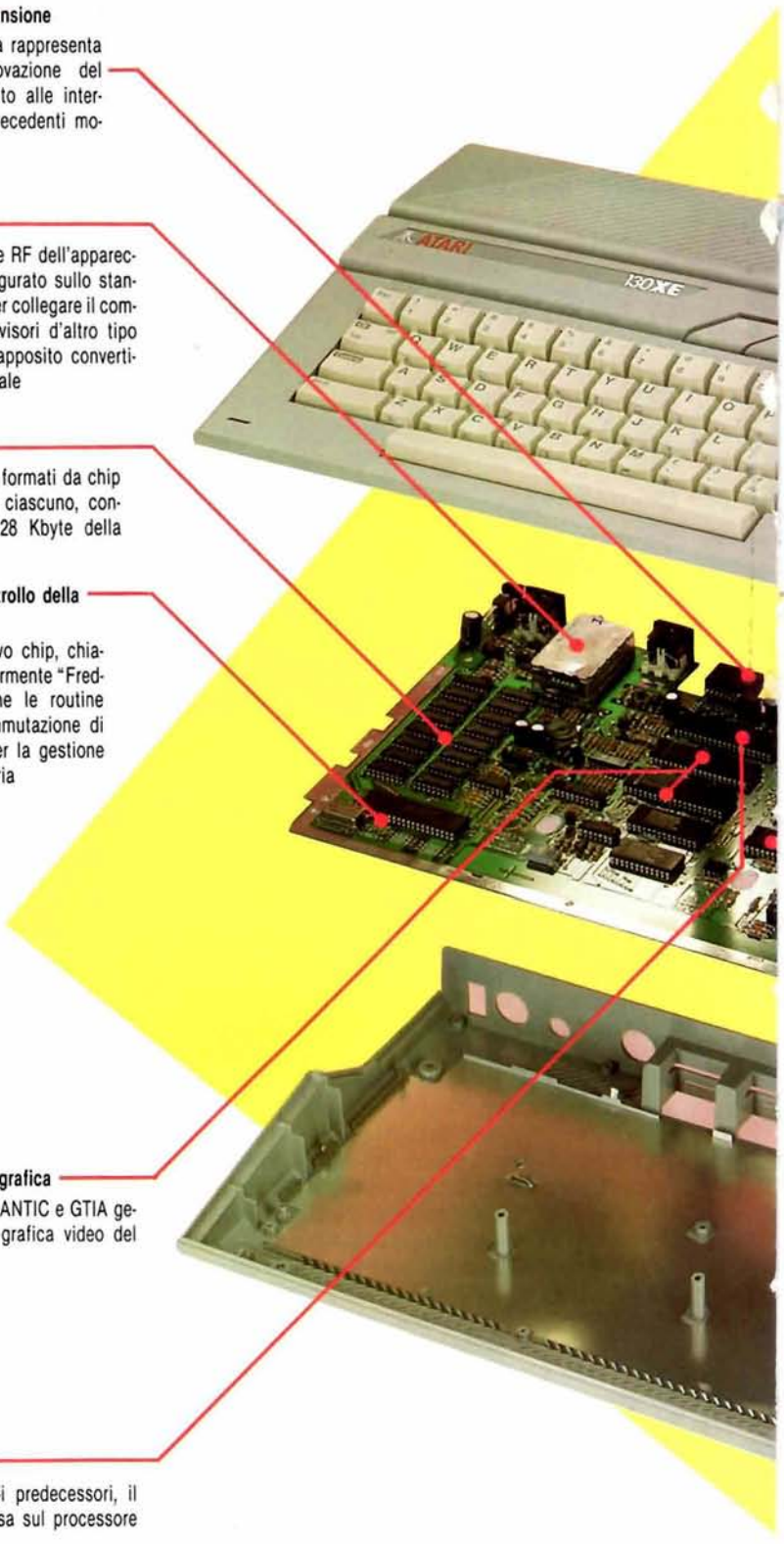
I tasti, ben sagomati e per niente rumorosi, facilitano l'uso della tastiera. La loro corsa è leggermente più lunga di quella delle macchine precedenti.

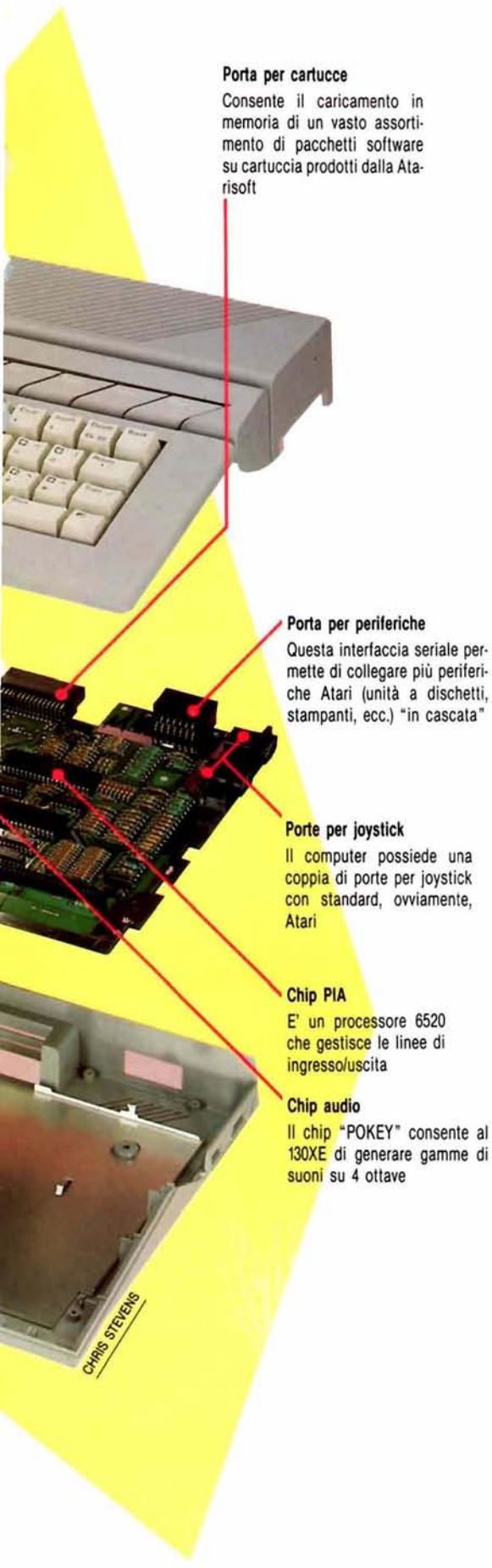
Come gli altri micro Atari basati sul 6502, il 130XE possiede 5 tasti funzione pre-definiti che, a differenza dei modelli precedenti, sono collocati sopra la tastiera e non alla sua destra. Il colore del loro rivestimento plastico è identico a quello della custodia mentre la tradizionale forma quadrata è stata sostituita da un'originale sagomatura a parallelogramma.

I tasti costituiscono un notevole miglioramento rispetto a quelli metallici della serie XL, che provocano una sensazione di scarsa affidabilità e funzionalità.

Collegamenti d'interfaccia

Le interfacce, situate sul retro e sul fianco destro del nuovo Atari, non offrono grandi sorprese. Lateralmente c'è una coppia di porte per joystick di tipo-D a 9 piedini, uno standard Atari oggi adottato da quasi tutti gli home computer. Sul retro, la porta seriale di controllo a 16 piedini permette di collegare "in cascata" le varie periferiche Atari (registratori a cassette, unità a dischetti e stampanti).
Sulla destra, alloggiati nella custodia, ci sono l'interfaccia d'espansione e il connettore per le cartucce.
Quest'ultimo consente di innestare sul computer una gran varietà di cartucce per giochi ad alta velocità della Atarisoft (quali *Pacman* e *Galaxian*, i due pacchetti software di maggior successo della compagnia).
La porta d'espansione si differenzia da quelle usate sui primi computer Atari, che possedevano un bus d'espansione con connettori a pettine da 50 canali.
Il 130XE, infatti, possiede un bus a 14 canali molto più piccolo e simile a un connettore per cartucce.
Le altre interfacce disponibili sono una porta per monitor a video composito, una presa per televisore e una comune presa di alimentazione tipo Atari.
Per contenere i costi di produzione, l'apparecchio viene commercializzato con un modulatore RF interno in versione unica, adatto ai televisori USA. In Europa lo standard è diverso, quindi può essere necessario un piccolo circuito per trasformare i segnali nel formato adatto. La Atari può vantare un'ottima progettazione e un'elevata compatibilità ma il successo della serie XL non è stato pari alle aspettative.
Il maggior punto di forza del 130XE, che lo pone al di sopra dei suoi predecessori, resta tuttavia la grande capacità di memoria abbinata ad un basso prezzo. Dato che un microprocessore ad 8 bit può indirizzare solo 64 Kbyte di RAM, per allocarne una quantità doppia il computer adotta una tecnica chiamata "commutazione di banco", che gli consente di aprire una "finestra" di 64 Kbyte sull'intera memoria di 128 Kbyte. Benché questa tecnica non sia quella ottimale e le istruzioni necessarie per passare da un banco RAM all'altro rallentino la ricerca delle informazioni, essa consente tuttavia l'indirizzamento di una quantità di memoria superiore di quella altrimenti possibile.
La commutazione di banco, comunque, è molto più usata di quanto comunemente si creda. Anche l'Oric, l'Atmos e il Commodore 64 possiedono una RAM interna superiore ai normali 64 Kbyte e ricorrono a questa tecnica per utilizzare appieno la memoria disponibile.
Un secondo punto di forza dell'Atari 130XE è il prezzo: l'apparecchio è più costoso dello Spectrum Sinclair e dell'Electron Acorn, ma più economico del Micro BBC e del Commodore 64. Per poter vendere un apparecchio a





128 Kbyte ad un prezzo così competitivo, la Atari ha dovuto tagliare drasticamente i costi di produzione. Ciò è stato possibile perché il 130XE è essenzialmente una macchina derivata da modelli precedenti e non ha quindi richiesto grosse spese di progettazione.

Taglio dei costi

La memoria è composta da 16 chip RAM da 8 Kbyte ciascuno, il cui costo di produzione, un tempo molto elevato, è oggi estremamente basso, grazie alle nuove tecnologie di fabbricazione.
Inoltre, per limitare il costo dell'apparecchio, la Atari ha deciso di ridurre al minimo il numero di componenti sulla scheda. Sebbene, per motivi di compatibilità, sia stato necessario mantenere molti dei chip attualmente presenti nei computer della serie XL, la scheda a circuiti stampati conserva una configurazione razionale e presenta un numero di componenti molto inferiore rispetto a quello di altri apparecchi con una memoria assai più limitata.
Infine, per la produzione del 130XE, la Atari ha installato catene di montaggio robotizzate, nelle quali vengono eseguite automaticamente tutte le saldature dei componenti sulla scheda.
I chip ROM per il BASIC, per la grafica e per il suono, praticamente identici a quelli adottati sui precedenti modelli Atari, consentono al nuovo apparecchio ottime prestazioni grafiche e di suono. Il manuale d'uso è stato ampiamente revisionato (le precedenti versioni apparivano, in effetti, un po' troppo elementari, tanto che potevano sembrare destinate a bambini). Anche il manuale del BASIC è stato migliorato e corredato da un'appendice contenente informazioni di carattere tecnico.
Tuttavia, per una più completa descrizione del linguaggio, è ancora necessario acquistare il "manuale di consultazione BASIC" edito dalla stessa Atari.
Benché la Atari avesse urgente necessità di rendere più competitiva la serie dei suoi micro, la repentina comparsa del 130XE presenta aspetti misteriosi. L'apparecchio offre al programmatore 64 Kbyte di memoria aggiuntiva ma, pur essendo compatibile con i package Atari ancora in commercio, le applicazioni in grado di sfruttare interamente le sue capacità di memoria si sono fatte attendere a lungo. Di solito, il lancio di un apparecchio di questo genere prelude a un intervento più significativo sul mercato dei piccoli computer gestionali, ma i nuovi responsabili della Atari hanno smentito questa consuetudine.
Forse il vero motivo del precipitoso lancio del 130XE si spiega con l'intenzione della Atari di fare concorrenza al nuovo Commodore 128, un micro compatibile con il Commodore 64 e dotato di memoria aggiuntiva, anche se più costoso del 130XE.

ATARI 130XE

DIMENSIONI

mm. 350 x 233 x 63

CPU

6502C funzionante a 1,79 MHz

MEMORIA

128 Kbyte di RAM, 24 Kbyte di ROM

VIDEO

Modo testuale: 40 x 24 caratteri. Modo grafico: 320 x 192 pixel (alta risoluzione) con 256 sfumature di colore

INTERFACCE

Porta per cartucce, presa TV, presa per monitor a video composito, due porte per joystick, porta d'ingresso/uscita seriale, interfaccia d'espansione

LINGUAGGI DISPONIBILI

LOGO, FORTH, PILOT e BASIC versione Atari

TASTIERA

Tipo QWERTY, con 62 tasti di cui 5 di funzione pre-programmati

DOCUMENTAZIONE

Il manuale fornisce esaurienti informazioni sul BASIC Atari, benché il tono generale tenda ad un'eccessiva semplificazione. In appendice si trovano dati tecnici per interfacciare le varie configurazioni e per l'utilizzazione dei 64 Kbyte di RAM aggiuntivi

PRO

Il nome Atari è sempre stato una garanzia di serietà e i 64 Kbyte di RAM offerti dall'apparecchio, considerato il suo prezzo non elevato, rappresentano un'interessante attrattiva

CONTRO

Il pieno successo dell'apparecchio è fortemente limitato da una carenza cronica di software, specialmente di quello prodotto da ditte indipendenti. Inoltre si tratta di un rifacimento di un apparecchio precedente, più che di un computer nuovo, e ciò può pregiudicarne la vendita su larga scala

UNA CODA DI ELEMENTI



Occhiate inquisitrici

Botta e risposta

Ecco come il PROLOG risponde ai quesiti. Nell'esecuzione, la variabile X assume due valori, che vengono entrambi conservati perché il PROLOG tratta le variabili come strutture locali per ogni chiamata di una clausola

Il controllo, nei programmi PROLOG, non è sequenziale ma segue una struttura ad albero

Nei linguaggi quali il BASIC o il PASCAL il flusso di controllo è tipicamente sequenziale, ossia viene trasferito da un'istruzione alla successiva in ordine discendente (fatta eccezione per i cicli e le diramazioni GOTO). Nel PROLOG, invece, il flusso di controllo assume l'aspetto di una ricerca su di un albero, in cui le clausole del programma vengono esaminate dando priorità alla "profondità".

Per meglio comprenderne il funzionamento, conviene rappresentare il programma usando un diagramma ad "albero", nel quale lo scopo (proposizione) da dimostrare occupa la radice, mentre tutti gli obiettivi intermedi corrispondono ai punti di scelta (nodi) da cui traggono origine le diramazioni inferiori. Un albero di questo tipo può essere esaminato in molti modi, ma il PROLOG inizia il lavoro dalla prima diramazione di sinistra e procede verso il basso fin dove è possibile. Se incontra una diramazione, ne verifica la validità, contrassegna il sentiero intrapreso come "già percorso" e continua verso il basso. Quando raggiunge un nodo terminale, indietreggia al punto di scelta precedente, imboccando il ramo immediatamente a destra di quello appena esaminato, e riprende la verifica discendente sulla nuova diramazione. In questo modo il sistema esplora tutti i "rami" dell'albero nel tentativo di dimostrare la veridicità della proposizione iniziale.

Una simile ricerca permette di esaminare l'intera struttura, ma richiede molto tempo, cosicché il PROLOG adotta particolari tecniche che consentono di evitare una parte del lavoro. Come già accennato, una clausola del linguaggio afferma che lo scopo iniziale è "vero" solo se risultano vere tutte le sotto-proposizioni che ne provano la validità:

```
scopo:— subscopo1, subscopo2, subscopo3... ecc.
```

La precedente clausola appare più comprensibile se espressa così:

```
Se (IF) sottoscopo1 è vero
   e (AND) sottoscopo2 è vero
   e (AND) sottoscopo3 è vero
   e (AND) ecc.
allora (THEN) scopo è vero.
```

Poiché i diversi sottoscopi sono connessi da operatori AND, basta che uno solo di essi non sia dimostrabile per rendere falsa l'intera clausola. Il PROLOG, quindi, esamina i sottoscopi (nodi) dell'albero da sinistra a destra e interrompe la ricerca se incontra una condizione di falso.

La tecnica del percorso a ritroso adottata nei programmi PROLOG rende irrilevante l'ordine di scrittura del codice, quindi il flusso di controllo assume un'importanza minore di quella che ha in BASIC, consentendo al programmatore di concentrare tutta la propria attenzione sulla struttura logica del problema da risolvere. Tuttavia, l'importanza che il PROLOG attribuisce alle istruzioni "dichiarative" non significa che il programma non possa comportarsi in modo procedurale. La clausola PROLOG:

```
marziano(X):— numero di arti(X.7), numero di teste(X.2), può programmare in (X, cobol).
```

può essere interpretata nel modo dichiarativo: «X è un marziano se possiede 7 arti, 2 teste e può programmare in COBOL», oppure nel seguente modo procedurale: «Per dimostrare che X è un marziano, occorre prima provare che possiede 7 arti, poi 2 teste e infine che può programmare in COBOL».

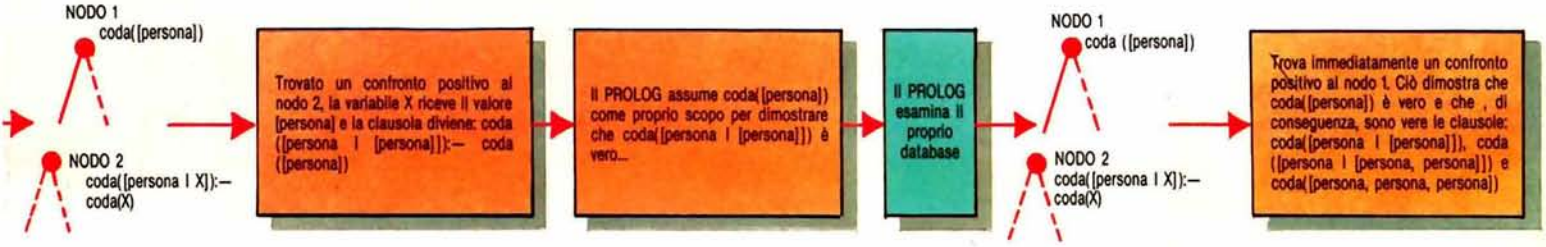
Liste e ricursione

Il PROLOG non è basato sui "tipi" come accade invece con tanti altri linguaggi, né appare altrettanto esigente per quanto riguarda i tipi di dato degli argomenti che compongono i suoi termini. Ad esempio, nel termine pred(Argomento), la variabile Argomento può essere di volta in volta definita come intero, come "atomo" (ad esempio: marziano, venusiano, d24 e così via), oppure come lista. Tuttavia il PROLOG elabora ciascun tipo di dato in modo diverso consentendo, ad esempio, l'elaborazione aritmetica dei numeri.

Chi ha una certa esperienza dei linguaggi LOGO e LISP ha già incontrato i dati del tipo "lista" e le particolari tecniche relative alla loro elaborazione. Una lista PROLOG è racchiusa tra parentesi quadre e i suoi elementi sono separati da virgole. Ad esempio: [mela, pera, banana] è una lista di frutti, [a, f, e, g, r, x] una di lettere e così via.

Il PROLOG permette di separare un elemento della lista alla volta rimuovendo il primo elemento. La notazione [Testa | Corpo] rappresenta una lista composta da un elemento Testa mentre il resto della lista è contenuto nella lista Corpo. Applicando la notazione | alla lista dei frutti si ottiene [mela | [pera, banana]]. Ciò significa che una lista può contenere altre liste come propri elementi. Tuttavia, se si considera la particolare lista [z80], composta da un unico elemento, e si applica la notazione "|", si ottiene [z80 | [ ]]. In tal caso, la lista contenuta nel corpo è [ ], ossia una lista vuota.

Nel PROLOG l'uso della ricursione è parte integrante del linguaggio. Come accennato più volte, una definizione ricursiva non è altro che una



KEVIN JONES

proposizione che si "autodefinisce", ossia l'equivalente di ciò che in linguistica si chiama "tautologia". Ad esempio:

```
coda([persona]).
coda([persona | X]):– coda(X)
```

Quando, come in questo caso, si hanno due o più clausole che possiedono la stessa testa, l'espressione prende il nome di procedura. La procedura coda possiede una clausola che definisce una coda come lista composta dal solo elemento persona e una seconda clausola che comunica che una coda può essere una lista la cui testa è l'elemento persona mentre il corpo è un'ulteriore lista chiamata X. Se si esamina la parte di destra di questa clausola, si nota che anche la lista X deve essere una coda. Assegnando al PROLOG il seguente scopo:

```
coda([persona, persona, persona])
```

al fine di verificare se la lista [persona, persona, persona] è una coda, per prima cosa il PROLOG cerca all'interno del proprio database una clausola che dimostri la veridicità dello scopo. La prima clausola è: coda([persona]) ma non viene trovata una corrispondenza, poiché le liste non sono uguali, quindi il PROLOG passa alla clausola successiva: coda([persona | X]):– coda(X). Poiché anche questa dimostra lo scopo, il PROLOG assegna alle variabili i seguenti valori:

```
coda([persona | [persona, persona]]):–
coda([persona, persona]).
```

Per provare che lo scopo della testa è vero, vanno dimostrati i relativi sottoscopi. Quindi il PROLOG assume coda([persona, persona]) come proprio scopo e inizia l'esame "discendente" delle clausole alla ricerca di un confronto positivo. In questo caso, il confronto con la seconda clausola dimostra la validità dello scopo, fornendo come risultato:

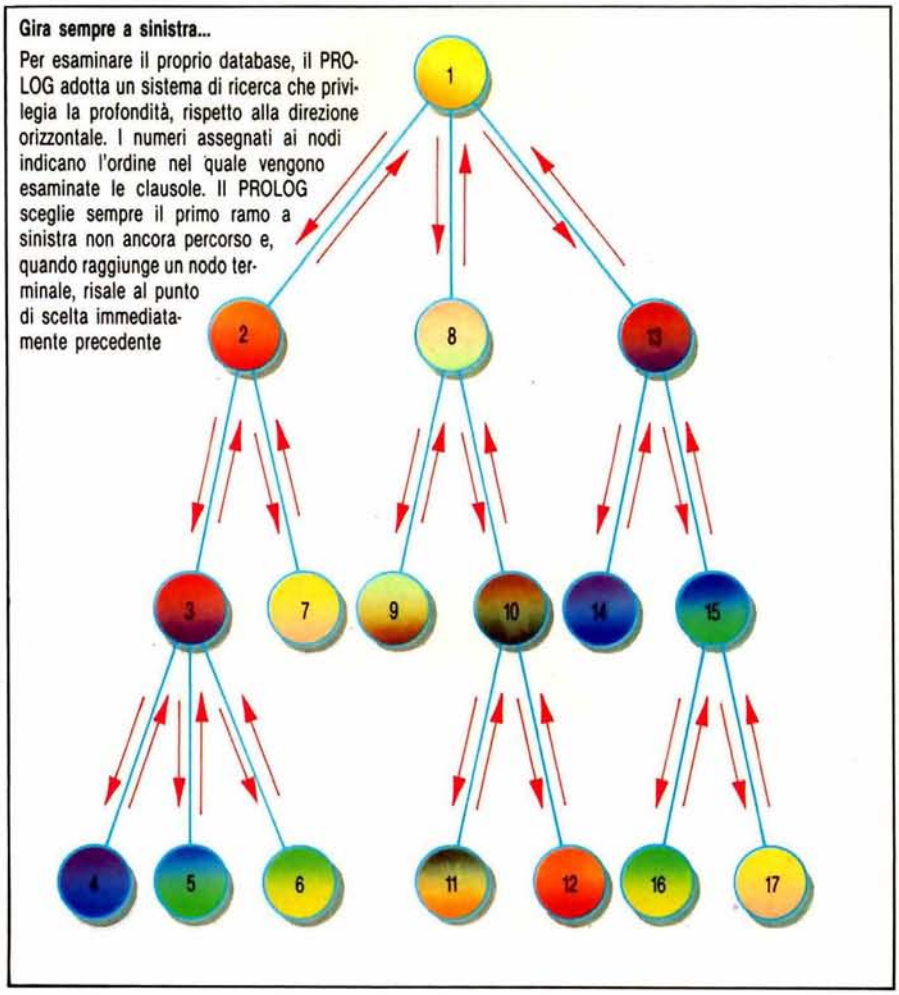
```
coda([persona | [persona]]):– coda([persona])
```

Si noti che la variabile X, definita in precedenza come [persona, persona], in questo caso contiene il valore [persona]. Quindi è stato mantenuto il primo valore di X. Ciò è possibile perché le variabili impiegate in ciascuna chiamata di una clausola sono locali; quindi ogni chiamata di coda ([persona | X]) usa una variabile unica e indipendente. Il PROLOG infatti non possiede variabili di tipo globale.

A questo punto, per provare la validità della testa dello scopo, occorre dimostrare il sottoscopo che precede il simbolo :–, ossia coda([persona]). Esaminando ancora la clausola di base, si ottiene un confronto positivo con la clausola coda ([persona]): questo risultato dimostra che la validità di coda([persona, persona]), ossia il nostro precedente scopo, e, di conseguenza, dello scopo iniziale coda([persona, persona, persona]). La procedura coda evidenzia alcune significative peculiarità del PROLOG. Ad esempio, che l'ordine delle clausole ha una grande importanza (si provi a invertire l'ordine delle due clausole della procedura e si osservi cosa accade quando si cerca di dimostrare uno scopo). Inoltre mostra come l'introduzione di nuove clausole equivalga all'impiego di procedimenti alternativi per dimostrare la validità di uno scopo: nel nostro caso, la presenza di una OR logica tra le due clausole. Quindi in PROLOG non occorre un operatore OR, sebbene esso sia previsto in molte versioni del linguaggio.

Gira sempre a sinistra...

Per esaminare il proprio database, il PROLOG adotta un sistema di ricerca che privilegia la profondità, rispetto alla direzione orizzontale. I numeri assegnati ai nodi indicano l'ordine nel quale vengono esaminate le clausole. Il PROLOG sceglie sempre il primo ramo a sinistra non ancora percorso e, quando raggiunge un nodo terminale, risale al punto di scelta immediatamente precedente



KEVIN JONES

Nuovo Mondo I

Il listato completo e le modifiche necessarie per i vari home computer

Il listato qui a fianco (la prima di tre parti) è stato redatto sul Commodore 64 ma, salvo pochi passaggi, la versione BASIC è compatibile con quella di altri apparecchi. Per usare il programma sullo Spectrum occorre attenersi ai seguenti principi: 1) lo Spectrum accetta solo nomi di variabile formati da una singola lettera per le matrici e i contatori di ciclo FOR... NEXT (consultare la tabella di conversione a fondo pagina). 2) Lo Spectrum adotta un proprio sistema di gestione delle stringhe quindi LEFT$, RIGHT$ e MID$ vanno sostituite con variabili equivalenti (gli accorgimenti in tal senso sono riportati negli specchietti "Basic & Basic" che corredano i listati dei vari moduli). 3) Le istruzioni PRINT CHR$(147) vanno sostituite da CLS, mentre le linee del tipo:

<n. di linea> GET I$: IF I$ = "" THEN <n. di linea>

che istruiscono il programma ad attendere la pressione di un tasto, vanno sostituite con:

<n. di linea> LET I$ = INKEY$: IF I$ = "" THEN GO TO <n. di linea>

La seconda parte del listato completo, con altri utili suggerimenti, viene fornita più avanti.

VARIABILI SPECTRUM TABELLA DI CONVERSIONE

Equivalente Microsoft	Significato	Spectrum
TS(,)	Rendimento/Tipo dell'equipaggio	T(,)
WG( )	Livello delle paghe	W( )
CC( )	Conteggio dei tipi	Y( )
PA( )	Provviste acquistate	A( )
PC( )	Costi delle provviste	C( )
PN( )	Fabbisogno provviste	N( )
OC( )	Costi delle merci	O( )
OA( )	Quantità delle merci	G( )
HR( )	Marcatori di metà razione	H( )
RR( )	Marcatori di eventi	R( )
AO( )	Ammontare dei baratti	E( )
EQ(,)	Tassi di scambio	Q(,)
V1( )	Prezzi alla partenza	B( )
V2( )	Prezzi a fine viaggio	D( )
S1	Cont. calo rendimento	S
S3	Cont. riduz. velocità di stampa	S
S4	Cont. ritardo breve	J
S5	Cont. ritardo lungo	S

```
1 REM ********************
2 REM **  NUOVO MONDO   **
3 REM **    GIOCO DI    **
4 REM **    STRATEGIA   **
5 REM ********************
6 :
```

La prima sezione del programma inizializza le variabili e le matrici impiegate nelle sezioni successive

```
9  K$ = "   PREMI QUALSIASI TASTO..."
10 DIM TS(16,2): REM CATEGORIA/RENDIMENTO DELLA
   CIURMA
11 CN=0:     REM NUMERO COMPONENTI DELLA CIURMA
12 MO=2000: REM MONETE D'ORO INIZIALI
13 DIM WG(5): WG(1)=10: WG(2)=25: WG(3)=15: WG(4)=20:
   WG(5)=15: REM PAGHE
14 WT=0:     REM TOTALE PAGHE SETTIMANALI
15 CM=16:    REM NUMERO MASSIMO MEMBRI
16 DIM C$(5): C$(1)="MARINAI": C$(2)="MEDICI": C$(3)=
   "MECCANICI"
17 C$(4)="NAVIGATORI": C$(5)="CUOCHI"
18 DIM CC(5): REM CONTO PER CIASCUNA CATEGORIA
19 **** MATRICI DI APPROVVIGIONAMENTO ****
20 DIM U$(4):  U$(1)="KG.": U$(2)="KG.": U$(3)="KG.":
   U$(4)="BARILI"
21 DIM P$(4): P$(1)="ORTAGGI": P$(2)="FRUTTA": P$(3)=
   "CARNE": P$(4)="ACQUA"
22 DIM PA(4)
23 DIM PC(4): PC(1)=.5: PC(2)=1: PC(3)=2: PC(4)=.5
24 DIM PN(4): PN(1)=2: PN(2)=1: PN(3)=1: PN(4)=.5
30 DIM OA(6)
31 DIM O$(6)
32 O$(1)="BOTTIGLIE DI MEDICINALE":  O$(2)="ARMI":
   O$(3)="SACCHI DI SALE"
33 O$(4)="PEZZE DI STOFFA": O$(5)="COLTELLI": O$(6)=
   "MONILI"
34 DIM OC(6)
35 OC(1)=1: OC(2)=5: OC(3)=.2
36 OC(4)=2: OC(5)=.5: OC(6)=1
40 JL=8: REM DURATA DEL VIAGGIO
41 EW=0: REM SETTIMANE SUPPLEMENTARI
42 DIM RR(16)
43 REM INDICATORI PER STABILIRE SE L'EVENTO (N) SI
   E' GIA' VERIFICATO
44 RC=0
45 REM CONTATORE DI EVENTI GIA' VERIFICATI
46 RM=13
47 G$="N": REM INDICATORE DI BEL TEMPO CHE
   INFLUISCE  NEL FATTORE DI AMMUTINAMENTO
48 A$="N": B$="N"
49 DIM M(6): REM SEGNALA GLI EVENTI GIA'
   ACCADUTI
60 DIM T$(3): T$(1)="PERLE": T$(2)="MANUFATTI": T$(3)=
   "SPEZIE"
61 DIM V1(3): V1(1)=2: V1(2)=2: V1(3)=1
62 DIM V2(3): V2(1)=2+(INT(RND(1)*1)/2): V2(2)=2+
   (INT(RND(1)*3)-1)
63 VD(3)=2+(INT(RND(1)*1)/2)
64 DIMEQ(4,3)
65 EQ(1,1)=0.5: EQ(1,2)=0.5: EQ(1,3)=1
66 EQ(2,1)=5: EQ(2,2)=5: EQ(2,3)=10
67 EQ(3,1)=3: EQ(3,2)=3: EQ(3,3)= 6
68 EQ(4,1)=2: EQ(4,2)=2: EQ(4,3)= 4
69 DIMAO(3)
80 PRINT CHR$(147): S$="  COMMERCIO NEL NUOVO MONDO*":
   GOSUB 9100: PRINT
81 GOSUB 9200
82 S$="SEI IL CAPITANO DI UN VASCELLO.*": GOSUB 9100:
   PRINT
83 S$="INTENDI NAVIGARE VERSO IL NUOVO*": GOSUB 9100:
   PRINT
84 S$="MONDO. DI SOLITO IL VIAGGIO DURA*": GOSUB 9100:
   PRINT
85 S$="8 SETTIMANE, MA PUO' RICHIEDERE*": GOSUB 9100:
   PRINT
86 S$="PIU' TEMPO. DEVI INGAGGIARE LA*": GOSUB 9100:
   PRINT
87 S$="CIURMA, PAGARLA E ACQUISTARE*": GOSUB 9100:
   PRINT
88 S$="MATERIALE VARIO PER IL COMMERCIO.*": GOSUB 9100:
   PRINT
89 S$="POSSIEDI 2000 MONETE D'ORO.*": GOSUB 9100: PRINT:
   GOSUB 9200
90 S$="         BUONA FORTUNA!!!*": GOSUB 9100: PRINT:
   GOSUB 9200: PRINT
91 S$=K$: GOSUB 9100
94 GET I$: IF I$="" THEN 94
95 GOSUB 9200
```

Questa sezione del programma principale chiama le subroutine che permettono al giocatore di ingaggiare l'equipaggio, caricare sulla nave le provviste per il viaggio e acquistare le merci da barattare

```
500 GOSUB 1000
550 GOSUB 2000: REM APPROVVIGIONAMENTO
600 GOSUB 3000: REM ACQUISTO MERCI DI SCAMBIO
601 REM ROUTINE DI PROVA
602 FOR T=1 TO 16
603 IF TS(T,2)=100 THEN TS(T,2)=INT(RND(1)*100)+1
604 NEXT
610 PRINT CHR$(147)
615 S$="SEI PRONTO A INIZIARE*": GOSUB 9100
625 S$=" IL VIAGGIO.*": GOSUB 9100
```

```
630 GOSUB 9200
635 PRINT: S$="COMPOSIZIONE DELL'EQUIPAGGIO: *":
    GOSUB 9100
640 GOSUB 9200
645 FOR T=1 TO 5
650 IF CC(T)=0 THEN 670
655 PRINT C$(T);": ";
660 PRINT TAB(15);CC(T)
662 IF CC(T)=1 THEN PRINT " ": GOTO 668
664 PRINT"S"
668 GOSUB 9200
670 NEXT
674 GOSUB 9200
675 PRINT: S$="HAI LE SEGUENTI SCORTE: *": GOSUB 9100
680 GOSUB 9200
685 FOR T=1 TO 4
690 IF PA(T)=0 THEN 710
695 PRINT PA(T);U$(T);" DI ";
700 PRINT P$(T)
708 GOSUB 9200
710 NEXT
715 GOSUB 9200
720 PRINT: S$="HAI INOLTRE ACQUISTATO: *": GOSUB 9100
725 GOSUB 9200
730 IF OA(1)=0 THEN 740
733 IF OA(1)=1 THEN S$="BOTTIGLIA DI MEDICINALE *":
    GOTO 735
734 S$="BOTTIGLIE DI MEDICINALE*"
735 PRINT OA(1);TAB(8);: GOSUB 9100
736 GOSUB 9200
740 IF OA(2)=0 THEN 750
743 IF OA(2)=1 THEN S$="ARMA DA FUOCO*": GOTO 745
744 S$="ARMI DA FUOCO*"
745 PRINT OA(2);TAB(8);: GOSUB 9100
746 GOSUB 9200
750 IF OA(3)=0 THEN 760
753 IF OA(3)=1 THEN S$="SACCO DI SALE*": GOTO 755
754 S$="SACCHI DI SALE*"
755 PRINT OA(3);TAB(8);: GOSUB 9100
756 GOSUB 9200
760 IF OA(4)=0 THEN 770
763 IF OA(4)=1 THEN S$="PEZZA DI STOFFA*": GOTO 765
764 S$="PEZZE DI STOFFA*"
765 PRINT OA(4);TAB(8);: GOSUB 9100
766 GOSUB 9200
770 IF OA(5)=0 THEN 780
773 IF OA(5)=1 THEN S$="COLTELLO*": GOTO 775
774 S$="COLTELLI*"
775 PRINT OA(5);TAB(8);: GOSUB 9100
776 GOSUB 9200
780 IF OA(6)=0 THEN 790
783 IF OA(6)=1 THEN S$="MONILE*": GOTO 785
784 S$="MONILI*"
785 PRINT OA(6);TAB(8);: GOSUB 9100
786 GOSUB 9200
790 GOSUB 9200
792 PRINT: PRINT "RESTANO ";MO;" MONETE D'ORO"
796 GOSUB 9200
797 S$="PREMI UN TASTO: INIZIA IL VIAGGIO*"
798 GOSUB 9100
799 GET I$: IF I$=""THEN 799
800 WT=0
801 H$="N": REM INDICATORE RAZIONI DIMEZZATE
802 DIM HR(4): HR(1)=1: HR(2)=1: HR(3)=1: HR(4)=1
```

Il viaggio ha inizio. Il programma usa la variabile WK per tenere il conto delle settimane di viaggio

```
820 WK=1 : REM CICLO PRINCIPALE DEL VIAGGIO
825 GOSUB 4000: REM RAPPORTO SULLA SALUTE
    DELL'EQUIPAGGIO
830 GOSUB 4200: REM RAPPORTO SULLE PROVVISTE
835 GOSUB 4300: REM RAPPORTO SULLE ALTRE MERCI
840 GOSUB 9200: PRINT CHR$(147)
842 PRINT : PRINT : PRINT
843 S$="PROBABILMENTE IL VIAGGIO*": GOSUB 9100
844 PRINT "DURERA' ANCORA";JL-WK;" SETTIMANE"
845 GOSUB 9200
846 PRINT: S$=K$: GOSUB 9100
847 GET I$: IF I$="" THEN 847
850 GOSUB 5000: REM VERIFICA CONTEGGIO DELLA PAGA
855 GOSUB 5100: REM DISTRIBUZIONE DELLE RAZIONI
860 GOSUB 5500
861 REM GOTO GENERAZIONE EVENTI CASUALI
870 GOSUB 6500: REM ROUTINE DEGLI EVENTI PRINCIPALI
875 IF HR(3)=.5 AND RND(1)<.5 THEN PRINT CHR$(147):
    GOSUB 6050
878 REM APPARE L'ALBATRO SE A CORTO DI CARNE
879 GOSUB 7200
880 GOSUB 5300 : REM RAPPORTO DI FINE SETTIMANA
889 WK=WK+1: IF WK<=JL THEN 825
890 REM APPRODO AL NUOVO MONDO
891 GOSUB 10000
892 GOSUB 10070
893 GOSUB 10300
894 GOSUB 10500
999 END
```

Il programma principale termina qui. Il resto del listato è composto da subroutine che vengono chiamate dalla sezione principale

```
1000 PRINT CHR$(147): PRINT "FASE 1 - INGAGGIO CIURMA"
1010 PRINT "-------------------------"
1012 PRINT
1015 GOSUB 9200
1020 PRINT: PRINT"CATEGORIE DI EQUIPAGGIO: "
1025 GOSUB 9200
1030 PRINT
1040 PRINT"TIPO DESCRIZIONE      PAGA SETTIMANALE"
1050 PRINT"---- -----------      ----------------"
1060 PRINT"  1  MARINAI            10 MONETE D'ORO"
1070 PRINT"  2  MEDICI             25 MONETE D'ORO"
1080 PRINT"  3  MECCANICI          15 MONETE D'ORO"
1090 PRINT"  4  NAVIGATORI         20 MONETE D'ORO"
1100 PRINT"  5  CUOCHI             15 MONETE D'ORO"
1105 GOSUB 9200
1110 PRINT: PRINT: PRINT
1120 S$="QUALE TIPO SCEGLI (1-5)*": GOSUB 9100
1122 S$="(F = FINE INGAGGIO)*":  GOSUB  9100:  PRINT:
     INPUT I$
1125 CT=VAL(I$)
1128 IF LEFT$(I$,1)="F" THEN PRINT: PRINT"FINE
     INGAGGIO.": GOSUB 9200: GOTO 1310
1130 IF CT>0 AND CT<6 THEN 1150
1139 PRINT: PRINT
1140 PRINT I$;: S$="  NON E' UN TIPO PREVISTO!*":
     GOSUB 9100
1142 GOSUB 9200
1145 S$="RIPETI IL COMANDO...*"
1146 GOSUB 9100
1147 GOTO 1300
1150 PRINT: PRINT
1155 CN=CN+1: REM NUMERO DI COMPONENTI LA
     CIURMA
1156 TS(CN,1)=CT: REM.CATEGORIA SCELTA
1157 TS(CN,2)=100: REM RENDIMENTO
     INIZIALE
1158 WT=WT+WG(CT): REM TOTALE PAGHE
1159 CC(CT)=CC(CT)+1: REM CONTO CATEG.
     CIURMA
1160 S$="CIURMA ATTUALE: "
1170 FOR T=1 TO 5
1180 PRINT S$;CC(T);" ";C$(T);
1189 S$="                "
1190 NEXT
1195 PRINT: PRINT"TOTALE PAGHE SETTIMANALI "
     ;WT
1200 IF CN=CM-1 THEN PRINT: S$="ULTIMO MEMBRO*": GOSUB
     9100: GOTO1295
1202 IF CN=CM THEN PRINT : S$=" EQUIPAGGIO COMPLETO!!*":
     GOSUB 9100: GOTO1310
1295 REM
1300 GOTO 1015
1310 PRINT: S$=K$: GOSUB 9100: PRINT: GOSUB
     9200
1320 GET P$: IF P$="" THEN 1320
1999 RETURN
2000 REM *** APPROVVIGIONAMENTO ***
2005 PRINT CHR$(147): REM PULIZIA SCHERMO
2010 S$="FASE 2 - APPROVVIGIONAMENTO*"
2015 GOSUB 9100
2020 S$="------   ------------------*"
2025 GOSUB 9100
2030 GOSUB 9200: PRINT
2040 PRINT "HAI INGAGGIATO ";CN;" UOMINI."
2045 GOSUB 9200: GOSUB 9200
2050 FOR T=1 TO 4
2055 PRINT
2060 PRINT "PER OGNI SETTIMANA DI VIAGGIO "
2070 PRINT "CIASCUN MEMBRO DELL'EQUIPAGGIO "
2075 PRINT "HA BISOGNO DI ALMENO: "
2080 PRINT PN(T);" ";U$(T);" DI ";P$(T)
2085 PRINT "COSTO: ";PC(T);"MONETE D'ORO AL";
     U$(T)
2090 PRINT "PER OGNI SETTIMANA"
2095 GOSUB 9200: PRINT: GOSUB 9200
2100 PRINT"QUANTITA' DI ";U$(T);
     "(";P$(T);")"
2110 S$="DA ACQUISTARE*": GOSUB 9100
2120 PRINT
2130 INPUT I$
2140 PA(T)=VAL(I$): GOSUB 9200
2150 IF PA(T)>((CN*8*PN(T))-1) THEN 2260
2160 IF PA(T)=0 THEN PRINT"SE NON NE ACQUISTI":
     GOTO 2180
2170 PRINT"SE ACQUISTI SOLO";PA(T);U$(T);
2180 PRINT P$(T);",": GOSUB 9200
2190 PRINT "QUALCUNO PUO' SOFFRIRE LA ";
2200 S$="FAME"
2210 IF T=4 THEN S$="SETE"
2220 PRINTS$;"!": GOSUB 9200
2230 S$= "RIPETI L'ORDINAZIONE":
     GOSUB 9100
2240 INPUTP$: P$=LEFT$(P$,1)
2242 IF P$<>"S" AND P$<>"N" THEN 2230
2245 IF P$="N" THEN 2400
2250 PA(T)=0: T=T-1: GOTO 2410
2260 IF PA(T)*PC(T)>MO THEN 2270
2265 GOTO 2400
2270 S$="NON HAI DENARO SUFFICIENTE": GOSUB 9100
2290 GOSUB 9200
2300 S$="RIPROVA*": GOSUB 9100: PA(T)=0: T-1: GOTO
     2410
2400 MO=MO-(PA(Y)*PC(T))
2410 PRINT: S$="PROVVISTE FATTE *": GOSUB 9100
2412 GOSUB 9200
2415 FOR TT=1 TO 4
2420 PRINT PA(TT);U$(TT);" DI ";P$(TT)
2450 GOSUB 9200
2460 NEXT
2480 PRINT"DENARO RIMASTO = ";MO;" MONETE D'ORO"
```

```
2485 GOSUB 9200: GOSUB 9200
2490 NEXT T
2500 GOSUB 9201:  PRINT:  S$= "FINE APPROVVIGIONAMENTO*
     ": GOSUB 9100: GOSUB 9200
2510 PRINT: S$=K$: GOSUB 9100: PRINT: GOSUB 9200
2520 GET I$: IF I$="" THEN 2520
2999 RETURN
3000 REM **** FASE 3: MERCI DI SCAMBIO ****
3001 PRINT CHR$(147): REM FASE 3
3002 GOSUB 9200
3005 PRINT "     FASE 3 - ALTRE MERCI  "
3010 PRINT "     ------   -----------  "
3020 GOSUB 9200
3025 PRINT
3030 S$="DURANTE IL  VIAGGIO, POSSONO*":
     GOSUB 9100
3035 S$="OCCORRERE ALTRE COSE, AD ESEMPIO*":
     GOSUB 9100
3040 S$="MEDICINALI, MERCI DI SCAMBIO,*":
     GOSUB 9100
3046 GOSUB 9200
3050 S$="OPPURE... ARMI.*": GOSUB 9100
3055 GOSUB 9200: GOSUB 9200
3060 FOR T=1 TO 6
3065 PRINT
3070 PRINT O$(T);": "
3075 S$="COSTO IN MONETE D'ORO: *": GOSUB 9100
3080 PRINT OC(T);
3090 GOSUB 9200
3095 S$="VUOI ACQUISTARNE (S/N)*": GOSUB 9100
3110 GET P$ : IF P$= "" THEN 3110
3115 IF P$<>"S" AND P$<>"N" THEN 3095
3120 IF P$="N" THEN 3175
3125 GOSUB 9200
3130 S$="QUANTITA' DELL'ACQUISTO?*": GOSUB 9100
3135 INPUT I$
3140 TT=VAL(I$)
3145 IF OC(T) * TT > MO THEN 3150
3147 GOTO 3160
3150 S$="DENARO INSUFFICIENTE!!*": GOSUB 9100
3152 GOSUB 9200
3154 S$="DEVI MODIFICARE LA QUANTITA'...*": GOSUB
     9100
3155 GOSUB 9200: GOTO 3130
3160 MO=MO-(OC(T)*TT)
3165 OA(T)=TT
3170 GOSUB 9200
3175 PRINT
3176 PRINT" MONETE RIMASTE = ";MO
3200 GOSUB 9200: NEXT T
3205 GOSUB 9200: PRINT: PRINT
3210 S$="FINE DELLA FASE 3*": GOSUB 9100
3220 GOSUB 9200: PRINT
3230 S$=K$: GOSUB 9100
3240 GET I$: IF I$="" THEN 3240
3999 RETURN
```

Le seguenti subroutine vengono chiamate dal ciclo principale del programma per verificare le condizioni correnti della nave e dell'equipaggio e fornire al giocatore un rapporto settimanale

```
4000 REM RAPPORTO SULLA SALUTE DELL'EQUIPAGGIO
4010 PRINT CHR$(147)
4020 S$="     GIORNALE DI BORDO*": GOSUB 9100
4025 S$="     ------------------*": GOSUB 9100
4030 GOSUB 9200
4035 PRINT "ALL'INIZIO DELLA SETTIMANA";WK
4040 S$="LO STATO DELL'EQUIPAGGIO E'*": GOSUB 9100
4045 GOSUB 9200: PRINT
4055 PRINT
4060 FOR T=1 TO 16
4070 IF TS(T,1)=0 THEN 4110
4075 PRINT C$(TS(T,1));"      (";
4078 IF TS(T,2)=-999 THEN S$="MORTO !!!!!!!!)*":
     GOTO 4099
4080 IF TS(T,2)>75 THEN S$="IN PERFETTA SALUTE)*":
     GOTO 4099
4085 IF TS(T,2)>50 THEN S$="IN SALUTE)*": GOTO 4099
4095 IF TS(T,2)>25 THEN S$="MALATO ! )*": GOTO 4099
4098 S$="MOLTO MALATO  !!  )*"
4099 GOSUB 9100: GOSUB 9200
4110 NEXT T
4115 GOSUB 9200: PRINT
4119 WW=0
4120 FOR T=1 TO 5
4130 WW=WW+(CC(T)*WG(T))
4135 NEXT T
4140 S$="PAGA SETTIMANALE: *": GOSUB 9100
4145 PRINT WW;" MONETE D'ORO"
4150 GOSUB  9200
4155 WT=WT+WW
4160 S$="TOTALE PAGHE FINO AD ORA: *": GOSUB 9100
4165 PRINT WT;" MONETE D'ORO"
4170 GOSUB 9200
4175 PRINT "DENARO RIMASTO = ";MO;" MONETE D'ORO"
4180 PRINT: S$=K$: GOSUB 9100
4190 GET I$: IF I$="" THEN 4190
4199 RETURN
4200 REM RAPPORTO SULLE PROVVISTE
4205 PRINT CHR$(147)
4206 PRINT "ALL'INIZIO DELLA SETTIMANA ";WK: GOSUB
     9200
4210 S$="LA QUANTITA' DI PROVVISTE*": GOSUB 9100
4215 S$="RIMASTE E': *": GOSUB 9100
4220 PRINT: GOSUB 9200
4225 FOR T=1 TO 4
4226 IF PA(T)=0 OR PA(T)=-999 THEN 4240
4230 PRINT PA(T);U$(T);" DI ";P$(T)
4232 X=INT(PA(T)/(CN*PN(T)))
4235 PRINT " (SUFFICIENTI PER ";X;" SETTIMANE)"
4239 GOSUB 9200
4240 NEXT
4290 PRINT: S$=K$: GOSUB 9100
4295 GET I$: IF I$="" THEN 4295
4299 RETURN
4300 REM RAPPORTO SULLE ALTRE MERCI
4305 PRINT CHR$(147)
4306 PRINT "ALL'INIZIO DELLA SETTIMANA ";WK: GOSUB
     9200
4310 S$=" CI SONO ANCHE: *": GOSUB 9100
4320 PRINT : GOSUB 9200
4322 IF OA(1)=0 THEN 4332
4325 PRINT OA(1);: S$=" BOTTIGLIE DI MEDICINALE*":
     GOSUB 9100
4330 GOSUB 9200
4332 IF OA(2)=0 THEN 4342
4335 PRINT OA(2);: S$=" ARMI*": GOSUB 9100
4340 GOSUB 9200
4342 IF OA(3)=0 THEN 4352
4345 PRINT OA(3);: S$=" SACCHI DI SALE*": GOSUB 9100
4350 GOSUB 9200
4352 IF OA(4)=0 THEN 4362
4355 PRINT OA(4);: S$=" PEZZE DI STOFFA*": GOSUB 9100
4360 GOSUB 9200
4362 IF OA(5)=0 THEN 4372
4365 PRINT OA(5);: S$=" COLTELLI*": GOSUB 9100
4370 GOSUB 9200
4372 IF OA(6)=0 THEN 4380
4375 PRINT OA(6);: S$=" MONILI*": GOSUB 9100
4380 GOSUB 9200: PRINT
4382 PRINT "SONO RIMASTE IN CASSA ";MO;: S$=" MONETE
     D'ORO*": GOSUB 9100
4383 GOSUB 9200
4397 PRINT: S$=K$: GOSUB 9100
4398 GET I$: IF I$="" THEN 4398
4399 RETURN
5000 REM VERIFICA CONTEGGIO PAGA
5005 IF WT> MO THEN 5010
5008 GOTO 5099
5010 PRINT CHR$(147)
5020 PRINT: PRINT: PRINT
5025 S$="TRA L'EQUIPAGGIO CORRE VOCE*": GOSUB 9100
5030 S$="CHE NON HAI DENARO SUFFICIENTE*": GOSUB 9100
5035 S$="PER PAGARE GLI UOMINI ALLA FINE*": GOSUB 9100
5040 S$="DEL VIAGGIO.*": GOSUB 9100
5045 GOSUB 9200: PRINT
5050 S$="L'EQUIPAGGIO E' FURIOSO!!*": GOSUB 9100
5055 GOSUB 9200: PRINT
5060 S$="SPERIAMO CHE TU RIESCA A FARE*": GOSUB 9100
5065 S$="BUONI GUADAGNI!*": GOSUB 9100
5066 GOSUB 9200
5070 PRINT: S$=K$: GOSUB 9100
5080 GET I$: IF I$="" THEN 5080
5099 RETURN
5100 REM DISTRIBUZIONE RAZIONI
5103 PRINT CHR$(147)
5105 S$="  DISTRIBUZIONE DELLE RAZIONI*": GOSUB 9100
5106 S$="  ----------------------------*": GOSUB 9100
5107 GOSUB 9200: PRINT "SETTIMANA: ";WK: PRINT
5108 H$="N"
5110 FOR T=1 TO 4
5112 HR(T)=1
5115 IF PA(T) > 0 THEN 5180
5120 PRINT "NON C'E' PIU' ";P$(T);" !!!!!": GOSUB 9200
5130 S$="LA CIURMA SI STA INDEBOLENDO!!*": GOSUB 9100
5135 WF=10: GOSUB 9300
5139 GOTO 5290
5180 X=(PN(T)*CN)*(JL-WK+1)
5185 IF PA(T)<X THEN 5200
5190 GOTO 5270
5200 PRINT "STANNO CALANDO LE SCORTE DI ";P$(T)
5205 GOSUB 9200
5210 S$="VUOI DIMEZZARE LE RAZIONI DI*":  GOSUB 9100
5215 PRINT P$(T);" PER L'EQUIPAGGIO?"
5220 INPUT I$: I$=LEFT$(I$,1)
5221 IF I$<>"S" AND I$<>"N" THEN 5220:  REM RISPOSTA
     NON CORRETTA
5225 IF I$="N" THEN 5270
5230 HR(T)=.5: H$="S"
5240 WF=5: GOSUB 9300
5250 S$="LA CIURMA SI STA INDEBOLENDO!!*": GOSUB 9100
5270 X=PN(T)*HR(T)*CN
5272 IF X>PA(T) THEN X=PA(T)
5275 PA(T)=PA(T)-X
5280 IF PA(T)=0 THEN PA(T)=-999
5285 PRINT "DISTRIBUITI";X;U$(T);" DI ";P$(T)
5290 PRINT: GOSUB 9200: NEXT
5295 PRINT: S$=K$: GOSUB 9100
5298 GET I$: IF I$="" THEN 5298
5299 RETURN
```



IAN McKINNELL

Canali di pensiero

Selezione e invio di dati sui canali video e schermo dello Spectrum

Sullo Spectrum Sinclair la periferica d'ingresso (input) più usata è la tastiera, mentre quelle d'uscita (output) sono il televisore o la stampante ZX. Le periferiche, nel sistema Spectrum, vengono generalmente chiamate "canali" d'ingresso o d'uscita. Il video, ad esempio, viene indicato come un canale d'uscita. Lo scorrimento delle informazioni (dati immessi da tastiera o caratteri trasmessi allo schermo), viene chiamato "flusso". Quindi un flusso di dati può essere inviato a differenti canali hardware, purché la periferica selezionata sia in grado di gestirlo correttamente. I flussi e i canali vengono ampiamente descritti più avanti. Per il momento, per l'analisi del sistema di I/O dello Spectrum conviene iniziare dalla routine "output di un carattere", che si trova all'indirizzo &0010. Il carattere inviato a questa routine viene ritrasmesso al video o alla stampante, a seconda del canale precedentemente selezionato. La tabella a margine pagina contiene i codici a lettera singola relativi ai diversi canali dello Spectrum "non espanso" e i numeri di flusso che il SO associa ad essi. Si noti che, sul modello di Spectrum preso ad esempio, sono operativi solo i flussi compresi tra 0 e 3.

Per inviare un carattere a una data periferica, occorre innanzi tutto comunicare al SO dello Spectrum il tipo di flusso che si intende usare. Ad esempio, per visualizzare un carattere sullo schermo va selezionato (usando l'apposita routine ROM con indirizzo &1601) il flusso 2, che in tabella è associato al canale S. Prima della chiamata alla routine che provvede ad "aprire" il canale hardware appropriato, il numero del flusso va depositato nel registro A. Quindi, per aprire in uscita il canale S, si adoperano le seguenti istruzioni assembly:

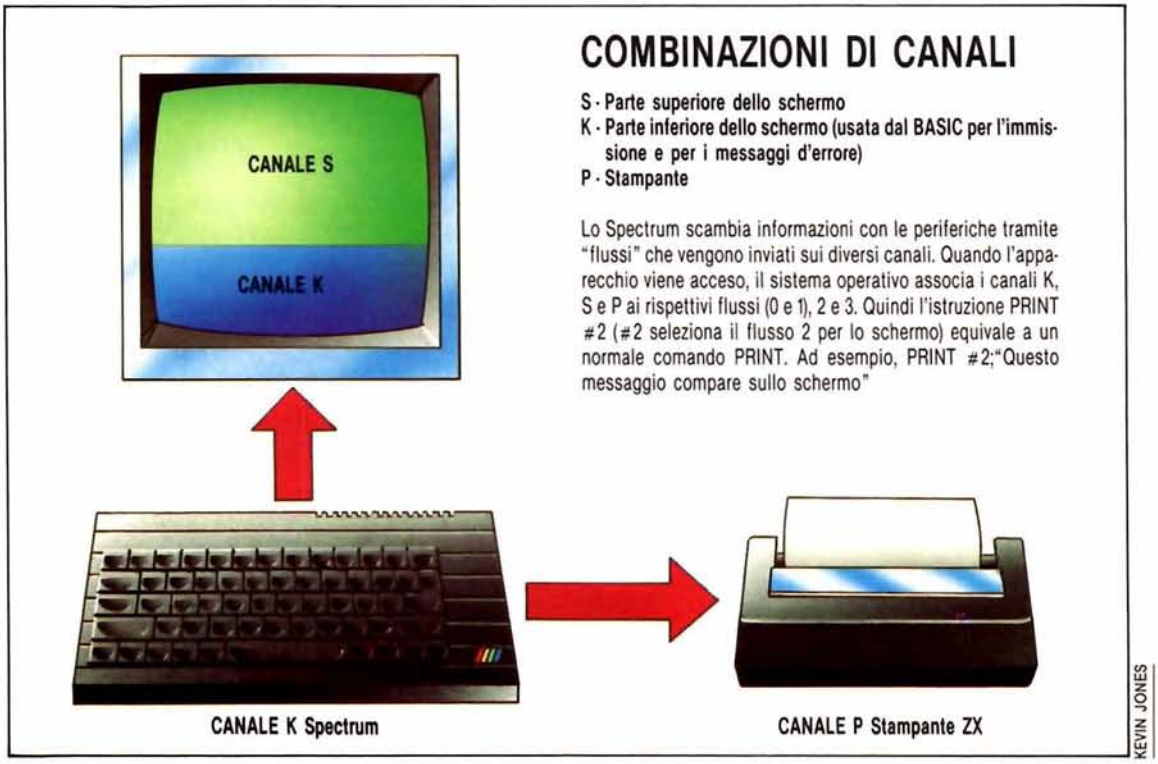
```
LD      A,2
CALL    &1601
```

Una volta aperto il canale, è sufficiente inserire il codice del carattere nel registro A ed eseguire un'istruzione RST, per chiamare la routine all'indirizzo &0010.

Tale chiamata somiglia per alcuni aspetti a quella della routine OSWRCH sul Micro BBC, ma si differenzia da essa in quanto l'indirizzo ROM viene chiamato direttamente e non attraverso un vettore. Il canale S opera sull'area dello schermo accessibile alle comuni istruzioni PRINT del BASIC. Tuttavia è possibile depositare caratteri anche nelle ultime due righe dello schermo, abitualmente usate dall'interprete BASIC per visualizzare i messaggi d'errore e i "prompt".

Per inviare un carattere a queste righe, occorre aprire il canale K con i comandi:

Flusso	Canale
0	K
1	K
2	S
3	P



COMBINAZIONI DI CANALI

S - Parte superiore dello schermo
K - Parte inferiore dello schermo (usata dal BASIC per l'immissione e per i messaggi d'errore)
P - Stampante

Lo Spectrum scambia informazioni con le periferiche tramite "flussi" che vengono inviati sui diversi canali. Quando l'apparecchio viene acceso, il sistema operativo associa i canali K, S e P ai rispettivi flussi (0 e 1), 2 e 3. Quindi l'istruzione PRINT #2 (#2 seleziona il flusso 2 per lo schermo) equivale a un normale comando PRINT. Ad esempio, PRINT #2;"Questo messaggio compare sullo schermo"

KEVIN JONES

```
LD      A,0
CALL    &1601
```

mentre il canale S della stampante viene aperto caricando nel registro A il valore 3.
Oltre ai caratteri standard, sui vari canali possono essere inviati anche caratteri di controllo. Naturalmente l'effetto ottenuto dipende dal tipo di canale selezionato, comunque ciò consente di usare istruzioni in codice macchina equivalenti ai comandi PRINT AT, PRINT INK, PRINT PAPER, ecc. Nella seguente tabella sono raccolti alcuni utili codici di controllo e la loro funzione quando vengono inviati ai canali K e S. Ovviamente, alcuni non producono alcun effetto, se trasmessi alla stampante ZX (attraverso il canale P).

Cod.	Parametri	Effetto
8	—	Cursore una posizione a sinistra
9	—	Cursore una posizione a destra
10	—	Cursore una posizione in basso
11	—	Cursore una posizione in alto
12	—	Cancellazione carattere
13	—	Codice di ENTER
16	n	INK n (richiede 1 byte extra)
17	n	PAPER n (richiede 1 byte extra)
18	n	FLASH n (richiede 1 byte extra)
19	n	BRIGHT n (richiede 1 byte extra)
20	n	INVERSE n (richiede 1 byte extra)
21	n	OVER n (richiede un byte extra)
22	nn	AT x,y (richiede 2 byte extra per le coordinate x e y)
23	n	TAB n (richiede un parametro extra)

I byte extra usati da alcuni codici di controllo corrispondono ai parametri contenuti nelle istruzioni BASIC equivalenti. Ad esempio, per eseguire il comando PAPER 3, al canale S vanno inviati i byte 17 e 3. Il seguente frammento di programma rappresenta il codice macchina equivalente a PRINT AT 10,10;"A".

```
3E02    10    ld    a,2       ; apre il canale S
CD0116  20    call  #1601     ; selezionando il flusso 2
3E16    30    ld    a,22      ; "token" per AT
D7      40    rst   #10       ; routine di output di un caratt.
3E0A    50    ld    a,10      ; coordinata y
D7      60    rts   #10
3E0A    70    ld    a,10      ; coordinata x
D7      80    rst   #10
3E41    90    ld    a,65      ; cod. ASCII per "A"
D7      100   rst   #10
C9      110   ret
```

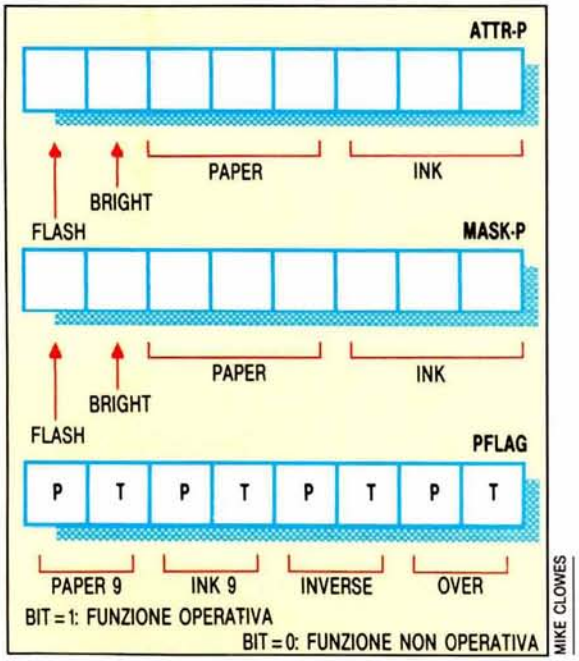
Si noti che, nella precedente tabella dei codici di controllo, non è presente il codice per CLS. Infatti, per cancellare lo schermo, va usata un'altra routine ROM, all'indirizzo &06DB. Come nel caso precedente, il canale S deve essere aperto prima della chiamata, ma se si desidera visualizzare un nuovo carattere, è necessario aprire il canale una seconda volta. La seguente routine cancella lo schermo:

```
3E02    10    ld    a,2
CD0116  20    call  #1601     ; apre il canale S
CDDB06  30    call  #06db     ; cancella lo schermo
3E02    40    ld    a,2
CD0116  50    call  #1601     ; riapre il canale S
C9      60    ret
```

Un'utile funzione prevista dalla routine Output di un carattere dello Spectrum standard consente di espandere e visualizzare per esteso le parole chiave del BASIC associate a particolari valori numerici (token). Se, ad esempio, è stato selezionato il canale S, P oppure K, le seguenti istruzioni:

```
LD      A,249
RST     &10
```

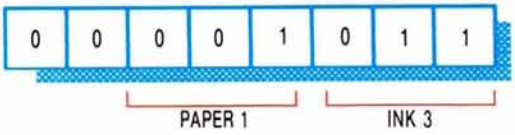
visualizzano la parola chiave RANDOMISE.
I comandi grafici PAPER, INK e BRIGHT, impartiti attraverso la routine Output di un carattere, hanno effetto solo sulla stringa di caratteri in uscita a cui si riferiscono: in questo caso vengono detti "informazioni temporanee di colore". Al contrario, il comando PAPER n, inserito al di fuori di un'istruzione PRINT, viene trattato come un'informazione di colore "permanente" e resta operativo finché non compare un secondo comando.



Il precedente diagramma mostra come le due variabili di sistema, chiamate ATTR-P e MASK-P, gestiscono differenti aspetti del display.
In ATTR-P, i tre bit che controllano in modo permanente i colori INK e PAPER forniscono i valori mostrati all'inizio della colonna successiva.
Se il bit associato a FLASH o BRIGHT contiene il valore 1, allora la funzione corrispondente risulta operativa. L'indirizzo in memoria di ATTR-P è 23693.
La variabile MASK-P si trova all'indirizzo 23694 e la presenza del valore 1 in un qualsiasi bit di questo byte garantisce che gli attributi per la relativa posizione di visualizzazione sullo schermo non vengano modificati dal contenuto di ATTR-P.
La terza variabile di sistema inserita nel diagramma è PFLAG. Questa si trova all'indirizzo 23697 e viene usata dal SO per indicare PAPER 9, INK 9, INVERSE e OVER.

Bit	Colore
000	Nero
001	Blu
010	Rosso
011	Magenta
100	Verde
101	Azzurro
110	Giallo
111	Bianco

Esistono infine altre due variabili di sistema, ATTR-T all'indirizzo 23695 e MASK-T all'indirizzo 23696, che funzionano in modo simile ad ATTR-P e a MASK-P, ma controllano i colori temporanei (quelli usati nelle istruzioni PRINT oppure quelli selezionati inviando codici di controllo allo schermo attraverso la routine RTS&10).
Usare ATTR-P e MASK-P per selezionare i colori è abbastanza semplice: è sufficiente manipolare il contenuto delle variabili di sistema modificando solo i bit necessari. Tale operazione può essere realizzata usando le istruzioni logiche AND e OR dello Z80. Quindi, per eseguire il comando di colore permanente PAPER1:INK3, occorre configurare la variabile ATTR-P nel modo seguente:



usando il segmento di programma:

```
LD      A,11
LD      (23693),A
RET
```

Naturalmente, come avviene con i comandi permanenti PAPER e INK del BASIC, i nuovi colori non influiscono sui caratteri grafici già stampati e lo schermo non visualizza il nuovo colore di PAPER finché non viene eseguito un comando CLS o un suo equivalente.
Dopo aver ricordato che le routine grafiche dello Spectrum fanno parte del programma dell'interprete BASIC, passiamo a esaminare il funzionamento di PLOT e DRAW. Queste routine, molto facili da usare, consentono di cambiare rapidamente il colore modificando il contenuto delle variabili ATTR-P e MASK-P nel modo prima descritto.
La prima, PLOT, viene chiamata all'indirizzo &22E5 e riceve le proprie coordinate dalla coppia di registri BC (B contiene la coordinata y e C la coordinata x). Quindi, per eseguire il comando PLOT 100,100 in un programma in codice macchina, si usano le istruzioni:

```
LD      B,100       ; coordinata y
LD      C,100       ; coordinata x
CALL    &22E5       ; esegue il comando
RET
```

Questa routine traccia un punto rosso sullo schermo e riporta ATTR-P allo stato originale, prima di restituire il controllo al BASIC.

```
3A8D5C   10    ld     a,(23693)  ; accoglie ATTR-P nel registro A
F5       20    push   af         ; e lo appoggia sullo stack
E6F8     30    and    248        ; azzera i 3 bit bassi
F602     40    or     2          ; bit 1 a 1 per selezionare il rosso
0664     50    ld     b,100      ; coordinata y
0E64     60    ld     c,100      ; coordinata x
CDE522   70    call   #22e5      ; chiama PLOT
F1       80    pop    af         ; ripristina ATTR-P
328D5C   90    ld     (23693),a  ; contenuto di ATTR-P
C9       100   ret
```

Esaminiamo adesso il canale K e le sue funzioni di uscita. Volendo, si possono visualizzare caratteri nella parte bassa dello schermo, anche se essi vengono poi "sovrascritti" quando il sistema operativo (o l'interprete) genera un messaggio d'errore. Comunque, per usare questo canale in ingresso, non c'è alcun bisogno di chiamare una routine ROM! La tastiera viene esaminata ogni 20 microsecondi e alcune variabili di sistema vengono modificate in base alla eventuale pressione dei tasti. Due di queste variabili sono particolarmente utili: LAST-K (con indirizzo 23560), che conserva il codice del carattere abbinato all'ultimo tasto battuto, e la variabile di sistema all'indirizzo 23556, che contiene il valore 255 se fino a quel momento non è stato premuto alcun tasto. La seguente routine introduce una pausa, che termina quando viene premuto un tasto (ossia quando il contenuto dell'indirizzo 23556 passa a 255), quindi carica il codice del carattere battuto nel registro A. A questo punto il valore registrato in LAST-K è quello del tasto premuto.

```
         10    ; GET-CHAR routine
3A045C   20    key  ld   a,(23556) ; routine ACCETTA UN CARATTERE
FEFF     30         cp   255       ; controlla se è premuto un tasto
28F9     40         jr   z, key    ; conserva il codice del carattere
3A085C   50         ld   a,(23560) ; carica LAST-K in A
CP       60         ret
```

La descrizione del canale K si conclude con una sintetica descrizione di altre tre variabili di sistema.

Variabile	Indirizzo	Descrizione
REPDEL	23561	Ritarda di un cinquantesimo di secondo la prima ripetizione del tasto. Modificando questo valore è possibile cambiare la durata del ritardo
REPPER	23562	Introduce un ritardo tra la seconda ripetizione del tasto e le successive. Il ritardo può essere modificato con un comando POKE
PIP	23609	Durata del "click" (segnale acustico associato alla battuta del tasto). I valori massimi producono un "beep" prolungato

Il terzo canale disponibile sullo Spectrum non espanso è il canale P, usato dalla stampante ZX e abbinato al flusso 3. Le istruzioni:

```
LD      A,3
CALL    &1601
```

aprono il canale e consentono di inviare i caratteri da visualizzare alla stampante collegata. Naturalmente il BASIC adopera ampiamente i vari canali: PRINT e LIST usano il canale S; LPRINT e LLIST il canale P e INPUT il canale K. In seguito è spiegato come i possessori di Spectrum possono creare nuovi canali, che permettono di collegare il computer ad altre periferiche (quali Microdrive, stampanti di diverso tipo e altre componenti hardware).

"Shadowfire"

Solo 100 minuti per salvare un ambasciatore prigioniero su di un'astronave

Anche un osservatore poco attento nota che i giochi per computer diventano ogni giorno più sofisticati. I loro creatori, abbandonato il filone dei giochi di tipo "arcade" e "adventure game" basati su testi di sapore vagamente medievale, sperimentano continuamente nuove strade. Questi giochi, pur presentando alcune caratteristiche dei pacchetti arcade, durano molto di più e richiedono spiccate doti strategiche e intuitive.

Nello *Shadowfire* viene narrata la cattura, da parte del Generale traditore Zoff, dell'Ambasciatore Kryxix, in possesso dei piani di una nuova astronave chiamata Shadowfire. Il giocatore deve riuscire, entro i 100 minuti a disposizione, a liberare l'ambasciatore prima che venga costretto a rivelare i piani. Il giocatore può essere aiutato da una squadra di sei personaggi, ciascuno caratterizzato da proprie abilità e debolezze.

Per introdursi nell'astronave di Zoff i soccorritori usano un "teletrasportatore", che solo il droide Manto è capace di attivare, raggiungendo per primo l'astronave nemica e azionando il fascio trasportatore che permette agli altri di seguirlo. Prima di sferrare l'attacco, è consigliabile assegnare a ciascun personaggio armi adeguate alle sue caratteristiche. Le armi vengono scelte da un "catalogo" visualizzato sullo schermo.

Lo *Shadowfire* adotta un originale sistema per muovere i personaggi, che consente loro di prendere le armi e combattere. Diversamente dai giochi d'avventura, lo Shadowfire permette di attivare tutte le funzioni di gioco posizionando il cursore su una serie di icone: una tecnica simile viene usata dal sistema operativo dell'Apple Macintosh.

Ad esempio, per armare di granate Zark, il comandante della squadra, occorre selezionare la sua icona. A questo punto l'immagine cambia e compare una rappresentazione grafica delle caratteristiche di Zark (forza, capacità offensive, ecc.) mentre nella parte destra dello schermo compaiono tre icone "monitor" relative al movimento, alle modalità di combattimento e all'armamento disponibile.

Posizionando il cursore su quest'ultima (con tastiera, joystick o penna luminosa), l'immagine cambia ancora e sullo schermo appaiono gli oggetti abbinati sia al personaggio che a una serie di icone d'"azione". Infine, per armare Zark di granate, si deve posizionare il cursore prima sull'icona che permette di prendere gli oggetti quindi su quella delle granate.

Una volta armata e teletrasportata sull'astronave, la squadra di salvataggio inizia la ricerca del Generale Zoff e dell'Ambasciatore Kryxix. All'interno l'astronave è suddivisa in stanze collegate da corridoi: alcune contengono armi o chiavi con le quali poter aprire le stanze chiuse, mentre altre nascondono reparti nemici da annientare per poter avanzare. Se il giocatore non possiede la chiave giusta, può chiamare in aiuto Sevrina, l'unico capace di far saltare le serrature.

Per utilizzare nel modo migliore i 100 minuti a disposizione, conviene sviluppare una strategia d'azione. Poiché ogni personaggio della squadra combatte meglio nelle situazioni di gioco a lui più congeniali, il giocatore deve cercare di schierare l'uomo giusto al momento giusto e nella posizione più favorevole.

La caratteristica più originale di *Shadowfire* è senza dubbio il controllo del cursore. Infatti il meccanismo di selezione delle icone consente al giocatore di reagire alle diverse situazioni molto più rapidamente di quanto accadrebbe se dovesse digitare comandi sulla tastiera. Tuttavia non basta avere buoni riflessi per portare a termine la missione, poiché per concludere il gioco, occorre anche rapidità nel prendere le decisioni e buone doti di stratega.

Shadowfire: Per il Commodore 64 e lo Spectrum Sinclair (le due versioni si trovano sulla stessa cassetta)
Editori: Beyond Software (GB)
Autori: Steven Cain, Dave Colcough, Karen Davies, Graham Everett, John Gibson, Fred Gray, John Heap, Ally Noble e Colin Parrot
Joystick: Opzionale
Formato: Su cassetta



Missione di salvataggio

Le immagini mostrano tre dei personaggi che il giocatore può chiamare in suo aiuto durante il gioco. A sinistra, Torik si trova nel modo "movimento" e le icone nel riquadro di sinistra rappresentano le direzioni consentite. Al centro, Zark Mondor si trova nel modo "combattimento". Le icone dei suoi attaccanti compaiono nel riquadro di sinistra, mentre i movimenti possibili sono visualizzati in quello centrale. Lo schermo di destra riproduce il droide da combattimento Maul e l'inventario degli oggetti disponibili. Il riquadro centrale, nella parte inferiore dell'immagine, contiene gli oggetti già assegnati al personaggio, quello di sinistra gli oggetti disponibili nelle immediate vicinanze. Per impartire istruzioni al computer basta posizionare il cursore (una croce bianca) sull'icona desiderata e premere il pulsante di fuoco del joystick

Come e cosa insegnare ai computer?

I tentativi di realizzare computer che imparano come bambini

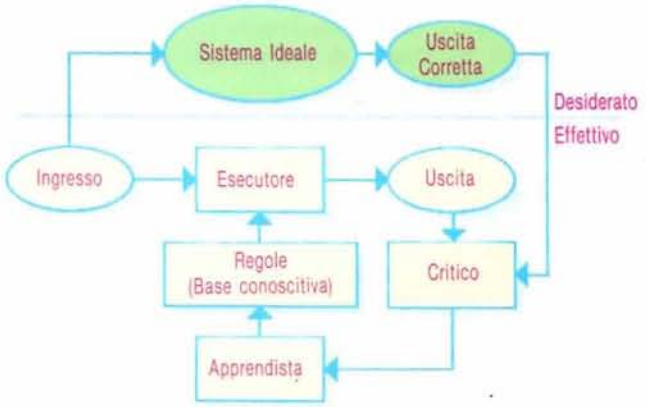
Si può affermare che un computer ha "imparato" qualcosa, se in un determinato arco di tempo è riuscito a migliorare le proprie prestazioni senza essere stato riprogrammato. Ciò implica l'adozione di un'unità di misura standard, con la quale poter misurare i progressi fatti dal computer.
Un algoritmo di apprendimento, deve raggiungere uno o più dei seguenti obiettivi:

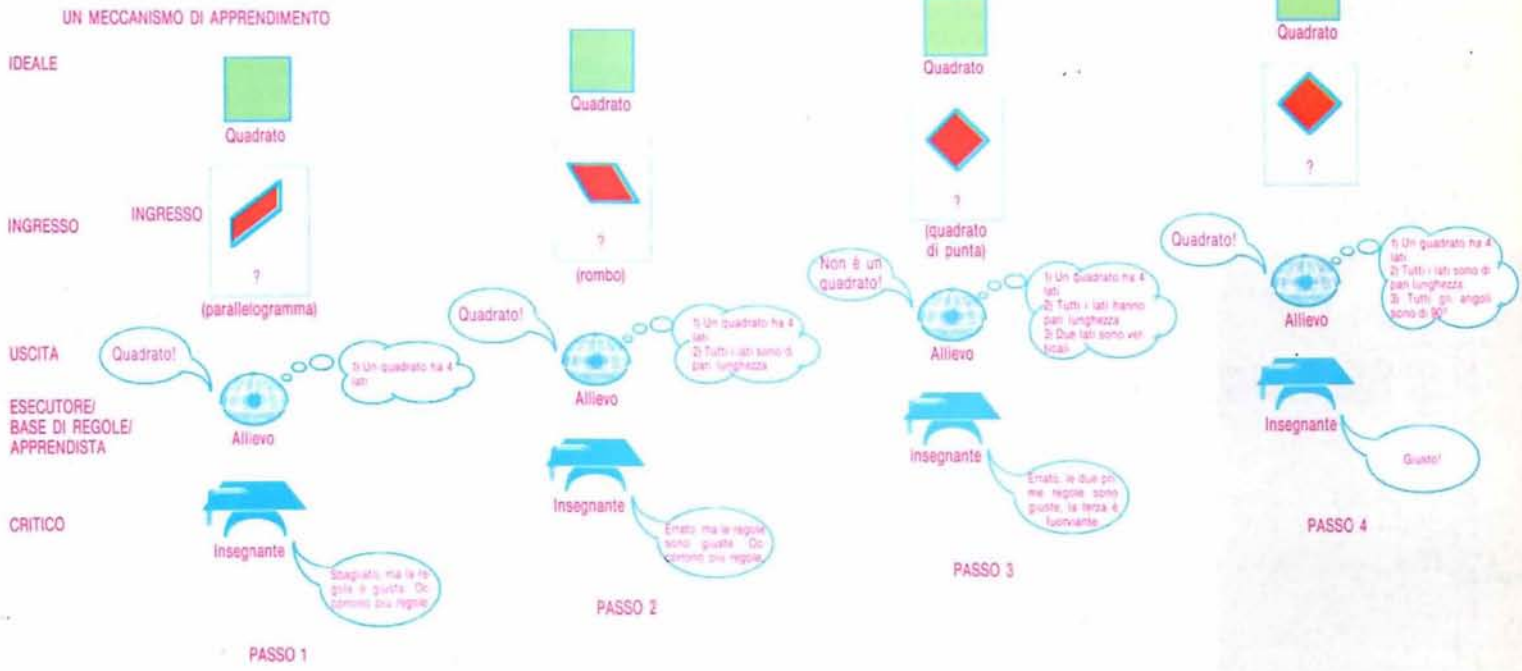
- Risolvere un numero maggiore di problemi
- Elaborare soluzioni più accurate
- Ottenere risultati ad un costo inferiore
- Semplificare la codifica della conoscenza

L'ultimo asserto è fondato sul fatto che una semplificazione della conoscenza codificata ha un notevole valore, anche se non rappresenta in realtà un miglioramento nelle prestazioni del computer. La situazione tipica è quella di un computer che parta da un insieme di regole e ne produca un altro equivalente, ma più comprensibile a chi usa il sistema. L'apprendimento da parte del computer può essere applicato in numerosi "domini" (vedere a pag. 1781) ma i sistemi di apprendimento di maggior successo sono quelli applicati alla classificazione di problemi. Scopo di questi sistemi è quello di esaminare i dati immessi per interpretarli ragionevolmente.

Intelletto critico

Il meccanismo di apprendimento nei bambini è ancora in gran parte ignoto. Tuttavia, alcuni psicologi hanno suggerito l'ipotesi che i bambini apprendano formando delle strutture di regole, dette "schemi", in base a un processo empirico (prova/errore). Gli schemi vengono poi ripetutamente messi alla prova, scartando quelli che forniscono risultati sbagliati. In genere questo processo viene pilotato da una presenza adulta (ad esempio il genitore o l'insegnante, con funzioni di "critico"), che ha già un'idea abbastanza precisa dell'obiettivo ideale cui tendere. Il critico assiste il bambino (l'apprendista) nella valutazione e nel miglioramento dell'insieme di regole possedute dal bambino, in modo che le sue prestazioni in un particolare compito possano via via raffinarsi. I sistemi di apprendimento dell'Intelligenza Artificiale tentano di riprodurre questo ambiente operativo





IAN McKINNELL

Sono stati ideati molti metodi per ottenere un miglioramento automatico delle prestazioni di un computer e i due più semplici fra questi, l'apprendimento a pappagallo (rote learning) e quello basato sull'aggiustamento di parametri, sono stati proposti da Arthur Samuel nei suoi studi sull'apprendimento delle macchine tramite il gioco degli scacchi. Ma, anche in precedenza, Rosenblatt e Selfridge avevano realizzato sistemi per il riconoscimento di configurazioni con capacità minime di apprendimento, come il Perceptron (vedere a pag. 1701).

Il metodo basato sull'aggiustamento di parametri non è altro, in realtà, che una tecnica di ottimizzazione: i coefficienti adoperati nelle valutazioni, inizialmente calcolati grossolanamente, vengono via via calibrati grazie all'esperienza fino ad assumere valori quantomai precisi. Questo metodo è stato studiato ed applicato a lungo anche in matematica, quindi nasconde ben pochi segreti. I sistemi basati sull'apprendimento "a pappagallo" sono caratterizzati invece da una totale assenza di creatività: ricorrono unicamente a tecniche di compressione sui dati per un più efficiente immagazzinamento di dati e non sono in grado di affrontare problemi mai incontrati prima.

Un sistema progettato per generare nuove conoscenze, e in grado di migliorare le proprie prestazioni, deve avere i seguenti requisiti:

- Un insieme di strutture di dati che rappresentino, in ogni momento, il livello di "competenza" raggiunto (le *regole*).
- Un algoritmo di utilizzo (l'*esecutore*) capace di usare convenientemente le regole per pilotare la risoluzione di problemi.
- Un modulo di feedback (il *critico*) in grado di confrontare i risultati via via ottenuti con l'obiettivo finale.
- Un meccanismo di apprendimento (l'*apprendista*) che adopera il feedback del critico per migliorare la definizione delle regole.

Il linguaggio descrittivo

Il metodo di rappresentazione adoperato per codificare la conoscenza del sistema ha almeno la stessa importanza dell'algoritmo di apprendimento. Prima di realizzare un sistema di apprendimento, quindi, è fondamentale assicurarsi che il linguaggio usato nel sistema sia capace di rappresentare le sottili distinzioni che possono presentarsi durante l'uso: una scelta non facile!

Supponendo per il momento di aver realizzato un siffatto linguaggio descrittivo, rimane il problema di automatizzare la generazione di accurate descrizioni al suo interno.

Un criterio adottabile può essere quello basato sulla ricerca, tra tutte le descrizioni disponibili, di quelle che risultano utili in un dato contesto. Un sistema di AI che si accinge ad apprendere può iniziare con una serie di parametri, necessari al processo di classificazione. Durante la fase di apprendimento, il sistema può generare e valutare nuove descrizioni combinando in vario modo i parametri originali, mantenendo le descrizioni ritenute utili ad una corretta classificazione e scartando le altre. Tuttavia, il numero di descrizioni generate, sintatticamente valide, assume presto valori astronomici. Inoltre, più espressive diventano le descrizioni, più complesso diventa il processo combinatorio. Va quindi cercato un metodo per pilotare la ricerca delle descrizioni, capace di ignorare la vasta maggioranza di quelle che risultano insignificanti.

Nel caso di dati "puliti" (ossia definiti con precisione senza zone di interferenza reciproca) il problema è minore: le difficoltà più grosse si incontrano quando i dati non sono classificabili così nettamente.

Vale la pena di esaminare in dettaglio uno dei vari sistemi realizzati in base ai principi esposti fin qui: il *BEAGLE* (Biological Evolutionary Algorithm Generating Logical Expressions). Questo è un sistema computerizzato che genera regole decisionali per induzione, partendo dalle informazioni contenute in un database. Un effetto secondario, da non trascurare, è l'individuazione della provenienza delle regole, in un sistema che si basa su di esse.

Il *BEAGLE* funziona sul principio della "selezione naturale", in cui le regole che non si adattano ai dati vengono rimosse e rimpiazzate da varianti migliori o da nuove regole create combinando due regole diverse. Le regole non sono che espressioni booleane, rappresentate tramite strutture ad albero.

Il software originariamente consisteva di due programmi in PASCAL: *HERB* (Heuristic Evolutionary Rule Breeder) e *LEAF* (Logical Evaluator And Forecaster).

Al programma *HERB* si devono fornire tre archivi di ingresso: uno contenente dati, uno i parametri di valutazione (payoff file) e uno le regole "vecchie". Il file prodotto in uscita contiene un insieme di nuove regole, buone almeno quanto quelle originarie. Il file dei dati contiene "informazioni di tirocinio", per le quali è nota la categoria di appartenenza. La matrice "payoff" definisce il valore (detto anche "costo") di classificazioni corrette o errate.

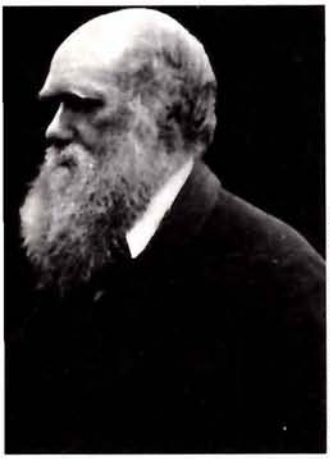
Il programma *LEAF* è più semplice: in ingresso viene fornito un solo file di dati, nello stesso formato del file di tirocinio, su cui sono applicate le regole definite in precedenza. In uscita, tra le altre cose, si può ottenere un elenco dei dati originali, ordinato in base alla possibile appartenenza di ciascun dato ad una categoria prefissata (definita nell'insieme delle regole).

L'algoritmo di apprendimento del *BEAGLE* consiste nella ripetizione della seguente procedura per un certo numero di "generazioni" (una generazione equivale ad un singolo passaggio attraverso i dati di tirocinio):

1. Valutazione di ciascuna regola su ciascun campione, in base alla matrice "payoff", assegnando un punteggio maggiore alle regole più efficaci.

Charles Darwin (1809-1882)

Le teorie di Darwin sull'evoluzione affermano che le specie si adattano all'ambiente e si migliorano tramite una selezione naturale: solo quelle più forti e che riescono ad adattarsi meglio sopravvivono per produrre nuove generazioni alle quali trasmettere le proprie caratteristiche. Principi analoghi sono stati applicati con successo all'Intelligenza Artificiale, facendo in modo che su un insieme di regole venga operata una specie di selezione naturale. Solo così il gruppo di regole "sopravvissute" garantisce un livello operativo efficace. Non è una coincidenza che il sistema "BEAGLE" abbia lo stesso nome della nave usata da Darwin nella sua spedizione alle Isole Galapagos



© BBC HULTON PICTURE LIBRARY

2. Classificazione delle regole in ordine decrescente di merito e rimozione della metà inferiore.
3. Riempimento della parte rimasta vuota con regole ottenute combinando coppie di regole, scelte fra quelle rimaste.
4. Creazione di varianti alle regole originali (ma non la prima in classifica) e successiva riorganizzazione (procedura TIDY) dell'insieme ottenuto, in preparazione alla generazione successiva.

La procedura TIDY rimuove eventuali sovrabbondanze sintattiche, quali le doppie negazioni e le espressioni costanti. L'albero di regole così "potato" è fondamentalmente lo stesso di quello originale, ma le regole risultano espresse più efficacemente.

Il criterio della generazione di nuove regole segue i principi di evoluzione espressi da Darwin: solo le regole più "forti" (ossia le più efficaci) resistono e rappresentano i "genitori" per la generazione successiva. Solo così l'insieme delle regole, durante le varie sessioni di apprendimento, cresce in modo ragionevole ed efficiente.

Nella prossima parte dedicata a questa serie viene proposta l'implementazione di un sistema BEAGLE in BASIC.

PREVISIONI DEL TEMPO

Un sistema di apprendimento può essere esemplificato come un meccanismo in grado di prevedere le condizioni meteorologiche di un giorno, basandosi su quelle del giorno precedente. La macchina deve "imparare" una regola che consenta una buona approssimazione. Il processo può iniziare fornendo al sistema una serie di descrizioni e dati relativi alle varie condizioni del tempo (quantità di pioggia, durata dei periodi di sole, massima velocità del vento e pressione atmosferica) registrata ad una data ora del giorno. La fase preliminare di apprendimento consiste nel fornire i dati storici di un particolare giorno (i cosiddetti dati di tirocinio) ed informando il sistema se il giorno successivo si sono avuti pioggia, vento, sole, ecc. Un record tipico può essere:

Pioggia 0
Sole 7.2
Vento 22
Pressione 1017

Il sistema deve generare regole (inizialmente anche secondo un criterio casuale) e adoperare i dati di tirocinio per verificarne la validità. Le regole possono essere conservate come singole componenti, ad esempio:

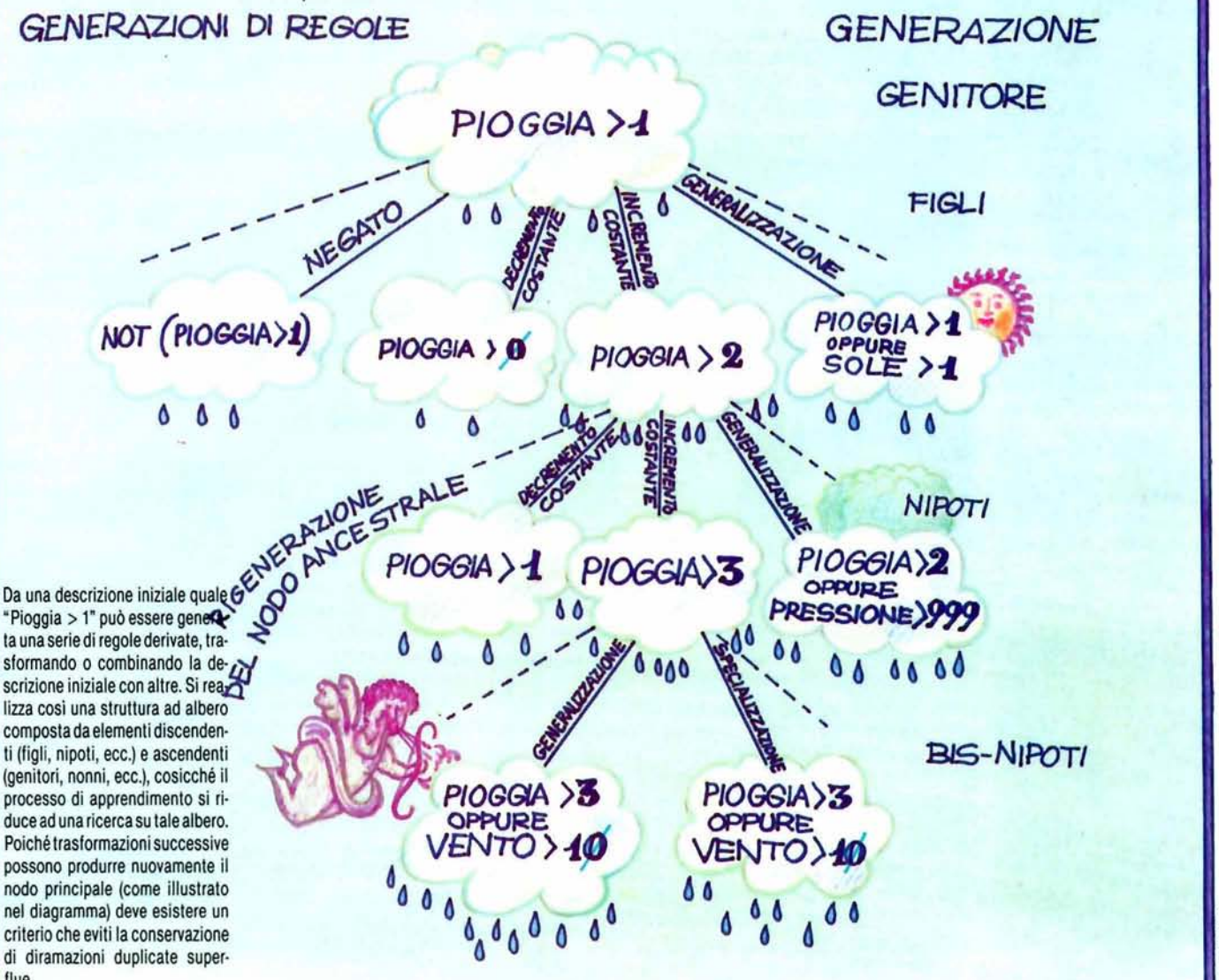
Particolarità Vento
Costanti 10

Operatori di confronto > oppure <
Operatori logici AND, OR e NOT

e combinate assieme in espressioni:

sole < 4 AND pressione < 1000

Dopo ciascuna sessione di tirocinio, le regole valide vengono mantenute, le altre eliminate e rimpiazzate da regole formate con combinazioni di quelle rimanenti. L'evoluzione dell'insieme di regole, proseguendo nell'apprendimento, dovrebbe raggiungere un livello ragionevolmente accettabile di previsione sulle condizioni meteorologiche dei giorni a venire



Da una descrizione iniziale quale "Pioggia > 1" può essere generata una serie di regole derivate, trasformando o combinando la descrizione iniziale con altre. Si realizza così una struttura ad albero composta da elementi discendenti (figli, nipoti, ecc.) e ascendenti (genitori, nonni, ecc.), cosicché il processo di apprendimento si riduce ad una ricerca su tale albero. Poiché trasformazioni successive possono produrre nuovamente il nodo principale (come illustrato nel diagramma) deve esistere un criterio che eviti la conservazione di diramazioni duplicate superflue

GABRIELLE IZEN

I dischetti e il CP/M

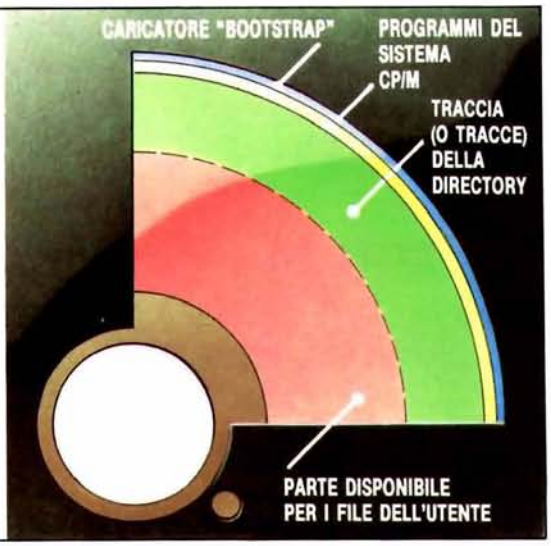
Dove risiedono e quale funzione hanno le varie parti del CP/M

Fin qui sono stati esaminati i vari comandi che permettono di creare, conservare, modificare e cancellare i file sui dischetti. Rimane un ultimo comando, non certo meno importante: quello che consente di copiare lo stesso CP/M su di un nuovo dischetto. Il CP/M, infatti, non può essere copiato come tutti gli altri archivi, poiché occupa una speciale porzione del dischetto, riservata appunto al sistema e normalmente non accessibile dall'utente.

L'importanza di poter copiare il sistema operativo su di un nuovo dischetto è fondamentale: non solo per effettuarne copie di riserva (nel caso venisse accidentalmente danneggiato il dischetto di lavoro), ma anche perché talvolta è necessario variare la configurazione del sistema operativo per adattarlo a speciali circostanze. Un esempio è dato dal *DrLOGO* sull'Amstrad: sul dischetto, assieme al linguaggio, viene fornita una speciale versione del CP/M, adatta alle funzioni richieste dallo stesso linguaggio.

TRACCE DA RICORDARE

Sebbene molte tracce in un dischetto CP/M siano disponibili per conservare dati e programmi, le tre più esterne sono riservate al sistema operativo. La traccia 0 contiene il programma che carica il CP/M in memoria all'accensione dell'apparecchio, mentre le altre due contengono i file dello stesso CP/M. La quarta traccia è riservata alla directory, ossia all'elenco dei File Control Block relativi a ciascun file archiviato sul dischetto



KEVIN JONES

Copiare il CP/M è abbastanza semplice: basta usare il comando transiente SYSGEN, che normalmente si trova sul dischetto di sistema. Questo programma visualizza una serie di quesiti, necessari per definire la nuova configurazione da dare al SO e per trasferire il nuovo CP/M sul dischetto di destinazione. Tuttavia, viene da chiedersi come mai sia necessario un comando a parte, perché i file e i comandi interni che compongono il CP/M non appaiano nella normale directory e, infine, come faccia il CP/M a leggere la stessa directory. Quanto segue tenta di rispondere a tali quesiti.

Il sistema operativo CP/M può essere suddiviso in tre parti: la CCP, il BIOS e il BDOS. La prima parte (Console Command Processor, ossia processore dei comandi da consolle) è quella più familiare, poiché gestisce le immissioni dell'operatore e le emissioni su schermo. In altre parole, rappresenta la "facciata" del CP/M.

Tuttavia, il vero lavoro è svolto interamente "dietro le quinte" dagli altri due moduli. La stessa CCP, per poter elaborare le immissioni e le visualizzazioni, ricorre al modulo BIOS (Basic Input Output System) che trasferisce i comandi in un buffer di memoria, detto appunto "buffer della CCP".

La CCP esegue quindi un controllo, per verificare se il comando impartito è tra quelli "interni", nel qual caso lo esegue immediatamente. In caso contrario la CCP cerca nella directory del dischetto un comando "transiente" il cui nome primario corrisponda a quello digitato e la cui estensione sia di tipo COM.

Localizzato il comando, il file ad esso associato viene caricato in memoria ed eseguito. Qualora anche questa ricerca fallisca, la CCP visualizza un messaggio per indicare che il file non è stato trovato. Per esser più precisi, il CP/M rivisualizza il comando impartito in caratteri maiuscoli (questa conversione viene sempre eseguita automaticamente dalla CCP su tutte le immissioni), seguito da un punto interrogativo.

Il modulo BIOS

Il BIOS consiste in una raccolta di routine (note anche come "driver per periferiche"), che si occupano di tutte le operazioni di ingresso/uscita del CP/M.

Il compito del BIOS è quindi quello di far funzionare correttamente la periferica alla quale si accede in un dato momento (sia essa la tastiera, il monitor, la stampante o i dischetti).

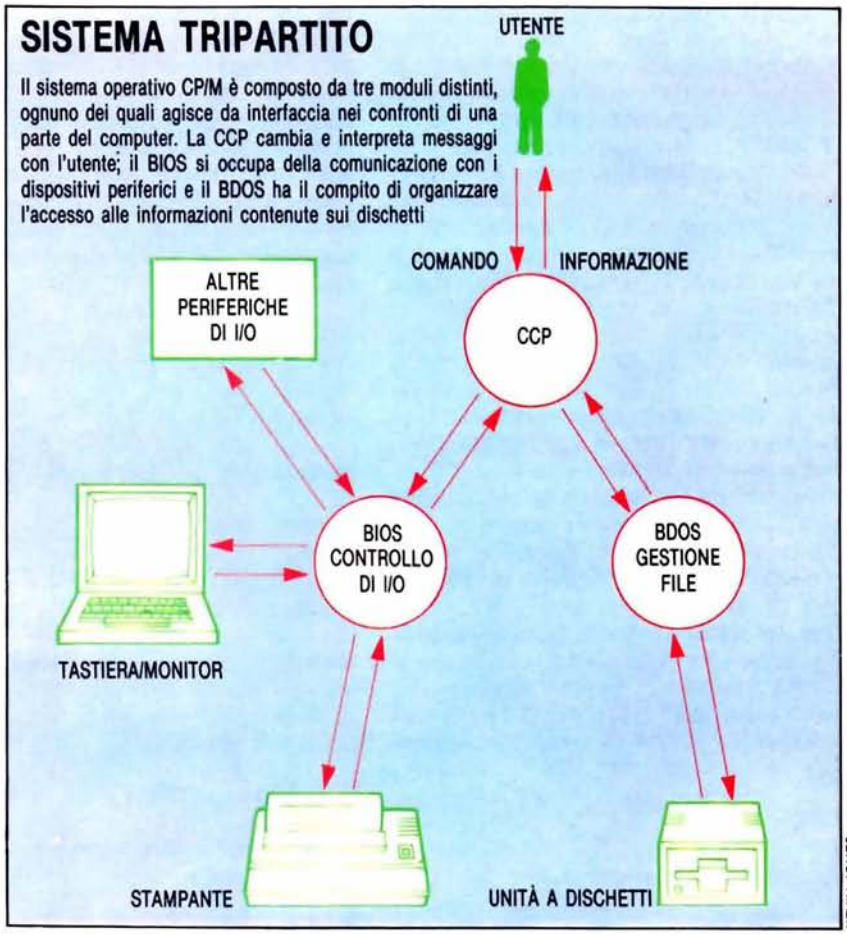
In effetti, il BIOS può essere considerato come l'interfaccia software tra i dispositivi abbinati al computer e il sistema operativo; tant'è vero che se occorre adattare il CP/M ad un nuovo apparecchio, è proprio sul BIOS che si deve intervenire (e i manuali della Digital Research spiegano come).

La terza parte del CP/M, il Basic Disk Operating System (detto anche BDOS), è il modulo che gestisce, sempre attraverso il BIOS, le operazioni sui dischetti. Senza questo valido aiuto il CP/M non potrebbe accedere in modo

organizzato alle informazioni conservate sui dischetti.
È il BDOS che gestisce l'uso e l'occupazione dello spazio sul supporto magnetico, creando e aggiornando di continuo la directory. Per esaminare a fondo il BDOS, però, occorre prima sapere come viene organizzato lo spazio su di un dischetto.

Settori Soft e Hard

La descrizione che segue riguarda principalmente i dischetti da 5 1/4 pollici, ma i principi sono identici per i floppy da 8".
I dischetti, in termini di hardware, contengono 40 tracce.
Nei primi computer la suddivisione delle tracce in "settori hard" era determinata dall'hardware (in particolare dal numero di fori disposti attorno al buco centrale). Il CP/M invece suddivide le tracce in "settori soft" (così chiamati in quanto è il software ad operare tale suddivisione). Ovviamente, essendo disposti ad anelli concentrici, i settori sono in numero maggiore verso il bordo esterno del supporto e in numero minore verso il centro. La maggior parte delle tracce di un dischetto viene adoperata per conservare informazioni.
Comunque, sui dischetti da 5 1/4 pollici, le tre tracce più esterne (dalla 0 alla 2) vengono riservate al CP/M. In esse trovano posto il "bootstrap loader" (ossia il programma che carica il sistema operativo in memoria quando si accende il computer) e i file del CP/M vero e proprio. È il comando SYSGEN che provvede a scrivere queste informazioni nelle apposite tracce.
I settori (o "record") del CP/M sono composti da 128 byte. I record possono essere raggruppati per formare un'"unità", a sua volta composta da un massimo di 128 record, cosicché si hanno dei "blocchi" da 16 Kbyte.
Salvo il caso in cui un file sia proprio minuscolo, esso non può essere contenuto in un solo record, né è possibile conservare sul dischetto (per ragioni pratiche) tutti i "blocchi" in ordine sequenziale. Il BDOS, dunque, deve tener conto di dove siano fisicamente collocati i vari blocchi e per far ciò ricorre a informazioni abbinate al nome del file nella directory, in ciò che viene chiamato "FCB" (ossia "File Control Block") di ciascun file. Nei dischetti da 5 1/4 pollici tutti gli FCB vengono conservati nella quarta traccia (quella, appunto, della directory) e il loro aggiornamento è compito del BDOS.
Ogni FCB è composto da 33 byte. Il primo contiene la lettera identificatrice del drive sul quale si trova il file; i byte 2-11 contengono il nome primario e l'estensione del file; i byte 12-14 sono usati per indicare le dimensioni del file; il 15° byte contiene il numero di record che compongono il file. I successivi 15 byte contengono informazioni relative all'occupazione, in termini di settori, del file sul dischetto. L'ultimo byte viene usato per indicare il numero di record, qualora il file abbia dimensioni superiori ai 16 Kbyte gestibili da un solo FCB.
All'apertura di un file, la CCP provvede a copiare il File Control Block in memoria. Ciò consente al modulo BDOS di aggiornarlo, modificando, ad esempio, le informazioni sulla lunghezza complessiva del file.
Al termine delle operazioni, ossia quando il file viene "chiuso", l'FCB viene riscritto nella directory, sovrapponendosi alla vecchia versione.
Dopo questi chiarimenti, è facile intuire il meccanismo adoperato quando viene impartito un comando sulla tastiera. La CCP interpreta la prima parola dell'immissione come nome del comando e, se esso non è residente in memoria, il nome è passato, tramite il BIOS, al BDOS (vedere a pag. 1784). Il BDOS cerca una corrispondenza nella directory e, se la trova, legge e copia l'FCB in memoria, ricavando da esso i dati sull'occupazione fisica del file sul dischetto.
A questo punto il BDOS carica in memoria, nel corretto ordine, i settori interessati e il comando viene eseguito.
Poiché spesso un comando è seguito dal nome di un file sul quale operare interventi, il BDOS si occupa in modo analogo di aprire, leggere (e anche scrivere se necessario) sia il relativo FBC sia il contenuto del file di dati.
Nell'ultima parte dedicata a questo sistema operativo viene descritta la mappa di memoria di un tipico sistema CP/M.



Effetti collaterali

Le operazioni procedurali, in PROLOG, sono possibili ma solo con qualche compromesso

Come si è visto, il PROLOG è sostanzialmente un linguaggio logico, pertanto operazioni procedurali (quali la lettura o la scrittura di un file oppure operazioni aritmetiche) mal si adattano ad una simile logica predicata. Per gestire queste situazioni, il PROLOG è dotato di alcuni predicati incorporati. Alcuni di questi funzionano tramite "effetti collaterali": Write(Elem), ad esempio, è un'espressione che funziona sempre ed ha utili effetti collaterali (come quello di emettere il valore di Elem sul corrente canale di uscita).
Tutte le immissioni ed emissioni, in PROLOG, sono ottenute tramite questi effetti collaterali. I programmatori più puristi, che preferirebbero una maggior "logica" in tali operazioni, in genere si scandalizzano per un simile compromesso. Tuttavia, il sistema funziona e non ve ne sono altri.
La scrittura di dati su file avviene in modo identico a quella su schermo. Il predicato tell(Nomefile) informa il PROLOG che tutte le successive scritture devono avvenire sul file specificato e told serve per chiudere il file.
In modo simile, see(Nomefile) estrae dati dal file e seen conclude tale operazione. Per inviare e ricevere valori ASCII, il PROLOG dispone rispettivamente di put(Codice) e get.
Per le operazioni aritmetiche esistono appositi predicati (in realtà vere e proprie funzioni) il cui risultato viene calcolato dall'interprete. Uno di tali predicati è is, che viene adoperato nel seguente modo:

```
C1 is C + 1
```

In BASIC ciò equivale a:

```
C1 = C + 1
```

Si noti che l'espressione C is C + 1, che in BASIC funziona perfettamente, in PROLOG non è consentita. Ciò dipende dal fatto che in quasi tutti i linguaggi (BASIC, FORTRAN, PASCAL, ALGOL e C) le assegnazioni sono "distruttive". In altre parole, il contenuto della variabile (posta nella parte sinistra dell'espressione) viene sostituito (e quindi "distrutto") con il valore della parte destra dell'espressione, e pertanto la stessa variabile può anche comparire nella parte destra.
In PROLOG, al contrario, non esistono assegnazioni, bensì "unificazioni". Vale a dire che se una variabile assume un dato valore, esso viene mantenuto in tutto il resto della clausola, cosicché scrivere: C is C + 1 sarebbe come voler assegnare due valori alla medesima variabile.
Il predicato incorporato not è usato per ottenere la negazione di un argomento. Scrivendo not(X) si ottiene un risultato vero se X è falso e viceversa (il funzionamento è identico al NOT booleano del BASIC). In PROLOG, tuttavia, per verificare not(X) occorre prima dimostrare se X è vero o falso e ciò viene fatto in modo alquanto semplice: una cosa è falsa se non può essere dimostrata per vera! Si supponga di avere il seguente programma PROLOG:

```
linguaggio(pascal,difficile)
linguaggio(cobol,difficile)
linguaggio(basic,facile)
linguaggio(prolog,facile)
```

in versione MICRO-PROLOG:

```
(linguaggio PASCAL difficile)
(linguaggio COBOL difficile)
(linguaggio BASIC facile)
(linguaggio PROLOG facile)
```

Questa è una semplice elencazione di alcuni fatti riguardanti i vari linguaggi di programmazione. Adesso si potrebbe chiedere:

```
linguaggio(basic,LIVELLO)
```

dove LIVELLO è una variabile. Il PROLOG risponde:

```
LIVELLO = facile.
```

Ma chiedendo:

```
linguaggio(basic,semplice)
```

Il PROLOG risponde con un bel "no". Ciò accade perché l'obiettivo richiesto non può essere provato (non è esplicitamente dichiarato nel database, né è derivabile da alcun altro fatto o regola). Il PROLOG, ovviamente, non conosce il significato delle parole e non sa, quindi, che "semplice" e "facile" possono significare la stessa cosa. Inoltre, esso considera il proprio database come un "mondo chiuso", in cui tutto quanto c'è da sapere è conosciuto: ciò che non esiste non è dimostrabile per vero e quindi è falso.
Questa caratteristica è una delle "debolezze" del PROLOG e deriva dal meccanismo usato dall'interprete nel processo di verifica logica. Tale processo è anche chiamato risoluzione e non è in grado di elaborare direttamente informazioni negative. Si consideri questo esempio:

```
not(A):— B, C, D.
```

L'espressione vorrebbe significare che A è falso se B, C e D sono tutti veri, ma non la si può usare in PROLOG, che non ammette fatti negativi come testa in una clausola.

Caratteristiche "illogiche"

Il PROLOG non è un linguaggio esclusivamente dichiarativo: si è già visto come l'ordine delle clausole possa influenzare il funzionamento del programma, specialmente nel caso della ricursione. In effetti, la già compromessa purezza logica del linguaggio viene ulteriormente inquinata dalla presenza delle cosiddette "caratteristiche illogiche" fail e cut. La prima di queste, fail, si spiega facilmente: è un predicato che non manca mai di "fallire" quando lo si incontra; ma è grazie a questa funzione così insolita che il PROLOG riesce a funzionare. Ritornando al programma sui linguaggi, si può formulare il quesito:

```
linguaggio(lingua,facile)
```

al quale il PROLOG risponde con:

```
Lingua = basic
```

e si ferma, poiché ha trovato una prima clausola nel proprio database che dimostra l'obiettivo richiesto e non procede oltre. Tuttavia, per essere esauriente, la risposta dovrebbe elencare tutti i linguaggi "facili". Si può forzare il PROLOG a produrre tale elenco aggiungendo una regola del tipo:

```
listafacili:— linguaggio(Lingua,facile),
write(Lingua),nl,fail
```

e, nella versione MICRO—PROLOG:

```
((listafacili)(linguaggio X facile) (PX) PP FAIL)
```

Adesso, digitando listafacili, il PROLOG risponde con:

```
basic
prolog
no
```

Tentiamo di comprendere l'accaduto. Per prima cosa il PROLOG cerca di dimostrare l'obiettivo listafacili verificando linguaggio(Lingua,facile). Viene trovato il fatto linguaggio(basic,facile), cosicché alla variabile Lingua viene assegnato il valore basic. Il primo sotto-obiettivo è raggiunto: si passa a quello successivo. Con Lingua che vale basic il PROLOG incontra write(basic). Siccome la write è sempre valida (visualizzando basic come effetto secondario) si passa all'ultimo sotto-obiettivo, ossia fail. Questo, per convenzione, fallisce e il PROLOG "indietreggia" e incontra nuovamente write. Questa (come accade con qualsiasi predicato di valutazione) non viene mai eseguita durante un percorso a ritroso, quindi il PROLOG indietreggia fino a linguaggio(Lingua,facile) e cerca una nuova via per dimostrare l'espressione.
Anche il predicato cut (abbreviato con il simbolo "!") è usato per controllare i percorsi a ritroso ma, contrariamente a fail, funziona sempre. L'effetto secondario di cut è che, nel percorso a ritroso, impedisce al PROLOG di indietreggiare oltre. Ciò, in pratica, blocca la scelta di valori per variabili selezionate in precedenza e, inoltre, impedisce che vengano considerate clausole alternative contenute nella stessa procedura.
Un importante impiego di cut è in congiunzione con fail, per garantire che una calusola fatta fallire deliberatamente non venga ripercorsa più volte. Usando "!" e fail si può definire un predicato mancante:

```
mancante(Clausola):— Clausola,!,fail.
mancante(Clausola).
```

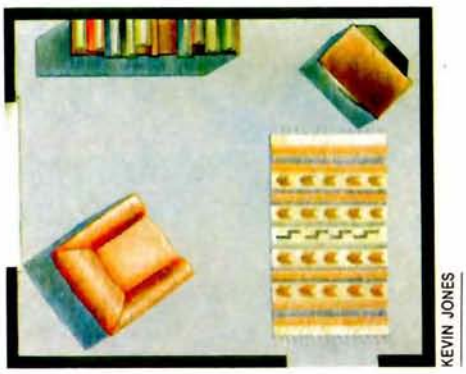
e in MICRO-PROLOG:

```
((mancante X)(?X)/FAIL
(mancante X)
```

che risulta vero se la clausola specificata è mancante dal database e falso in caso contrario. Adesso è posibile chiedere:

```
mancante(linguaggio(basic,facile)).
```

Per dimostrare mancante(linguaggio(basic, facile)), occorre prima provare linguaggio(basic,facile), che risulta vero poiché il fatto esiste nel database. Passando attraverso cut, che funziona sempre, si arriva a fail che provoca il fallimento dell'intera clausola.
Senza cut, il PROLOG indietreggerebbe per trovare un'ulteriore soluzione per Clausola. Invece la sua presenza contribuisce a far fallire immediatamente la ricerca, cosicché mancante(Clausola) diventa falso, in quanto Clausola è vera, e non si spreca tempo a cercare soluzioni alternative.
Questo esempio illustra un aspetto molto importante del PROLOG: il linguaggio può considerare parti dello stesso programma quali dati sui quali operare e tale caratteristica lo rende un linguaggio ideale per applicazioni di Intelligenza Artificiale.


KEVIN JONES

Da un luogo all'altro

In seguito, viene presentato un programma per far muovere un robot in una stanza come questa. Le varie posizioni nella stanza sono conservate nel formato posto(poltrona) ecc., mentre il robot si sposta usando la procedura vai(Posto1,Posto2). Questa implica la verifica di un sotto-obiettivo (che Posto1 esista e sia possibile usarlo come punto di partenza). Quali altri sotto-obiettivi occorre verificare per costruire un percorso da un luogo all'altro?

Prolog & Prolog

Il dialetto di PROLOG adoperato nel testo è il DEC-10 PROLOG (il nome deriva dal computer della Digital Equipment Corporation sul quale il linguaggio è stato creato) ed è internazionalmente riconosciuto come standard.
Sfortunatamente, una delle più diffuse implementazioni su microcomputer di questo linguaggio, ossia il MICRO-PROLOG, differisce notevolmente dallo standard.
L'altra più importante versione per micro, il PROLOG-1 (come la susseguente PROLOG-2) è invece conforme allo standard.
I programmi scritti in MICRO-PROLOG hanno un formato che ricorda molto le liste del LISP. Tuttavia, il programmatore può evitare la complessa notazione ricorrendo al programma di supporto, scritto in MICRO-PROLOG e chiamato "Simple", che fornisce una sintassi alternativa.
Esistono molte piccole differenze tra il MICRO-PROLOG e lo standard, che fanno talvolta sembrare i due linguaggi decisamente diversi tra loro. Ad esempio in MICRO-PROLOG il predicato è collocato come primo elemento di una lista. Si scrive quindi:

```
(casa_di Marte Marziano)
```

anziché:

```
casa_di(marte,marziano)
```

In MICRO-PROLOG, inoltre, viene eliminato l'uso dei simboli quali virgole e ":— ", cosicché una clausola del tipo:

```
casa_di(Pianeta,Creatura):—

nato(Creatura,Paese),

su(Paese,Pianeta)
```

diventa:

```
((casa_di x y)(nato y z) (su z x))
```

che può essere interpretata come una lista di tre sotto-liste, la prima delle quali è la testa della clausola. Ricorrendo al programma "Simple", la sintassi diventa senza dubbio più comprensibile:

```
x casa_di y if y nato z and z su x
```

che sarebbe ancor più comprensibile se, al posto delle lettere x, y, z, X, Y o Z (le uniche consentite, anche se è possibile combinarle con una cifra) si potessero adoperare nomi di variabili più significative.
Altre differenze del MICRO-PROLOG rispetto allo standard sono: l'uso di parentesi tonde (anziché quadre) per definire le liste; l'uso di spazi (e non di virgole) per separare gli argomenti e gli elementi delle liste; l'uso di "/" al posto di "!" per cut, e la definizione dei predicati incorporati in lettere maiuscole. Inoltre, alcuni predicati incorporati hanno nomi diversi dallo standard e, in certi casi, non operano in modo del tutto identico al predicato corrispondente.
Comunque, il MICRO-PROLOG contiene un'interessante caratteristica assente nel DEC-10 PROLOG: la possibilità di creare moduli di programma. Ciò significa che i moduli possono essere redatti separatamente e indipendentemente, senza preoccuparsi di eventuali casi di "omonimia" tra i nomi dei predicati. Questa caratteristica dota il MICRO-PROLOG di una certa strutturazione, anche se è ben lungi dall'essere un linguaggio veramente strutturato

Il Nimbus RM

Anatomia di un microcomputer a diffusione europea: tecnologie a confronto

Alla conquista di nuovi settori

Il Nimbus rappresenta una considerevole svolta progettuale per la Research Machines. Basato sul processore 80186, capace di indirizzare fino a 1 Megabyte di memoria, l'apparecchio prevede tutta una serie di coraggiose innovazioni, prima fra tutte la scelta di non andare ad infoltire la schiera degli IBM-compatibili



CHRIS STEVENS

Il nome della Research Machines Ltd non è molto noto al di fuori dell'ambito didattico delle scuole britanniche, che hanno adottato un vasto numero dei suoi microcomputer a 8 bit. In effetti, i due apparecchi 380Z e 480Z, entrambi basati sul chip Z80 e collegabili in rete, sono stati gli unici veri concorrenti del Micro BBC durante i primi sviluppi del progetto governativo britannico sull'informatica nelle scuole.

Nonostante gli apparecchi quali il 380Z siano stati considerati rivoluzionari al momento del loro lancio sul mercato, in seguito la domanda si è spostata verso una maggiore potenza elaborativa, ottenibile soltanto con micro a 16 bit. Di conseguenza, la Research Machines ha prodotto il Nimbus.

Ovviamente, dal tempo della prima apparizione del 380Z, l'industria si è sviluppata enormemente e il mercato è dominato dalla presenza del PC IBM e dei suoi innumerevoli "seguaci", più o meno compatibili con esso, e quasi tutti basati sul chip 8088. Questo è certamente più veloce rispetto al "glorioso" Z80, ma la Research Machines ha preferito discostarsi dalla tendenza generale di copiare il PC IBM, usando il più recente 80186, un vero microprocessore a 16 bit, capace di

eseguire un milione di istruzioni al secondo. Il risultato è un apparecchio che si lascia ampiamente alle spalle tutta la concorrenza. Benché il Nimbus non sia un IBM-compatibile, il chip 80186 è uno stretto parente dello 8088, tanto che su entrambi viene adoperato il sistema operativo MS-DOS. La tastiera del Nimbus è configurata per assomigliare a quella dell'IBM, possiede infatti 10 tasti funzione programmabili, una tastierina numerica e il tasto per i caratteri ALTernativi. Fisicamente, al tocco, i tasti sono gradevoli e affidabili.

La configurazione di sistema qui presentata è la PC2, dotata di due drive per dischetti da 3 1/2 pollici formato Sony, ognuno con una capacità di 720 Kbyte. La versione PC1 prevede una sola unità a dischetti.

Anche la scelta di impiegare queste unità a dischetti, assieme ad altre caratteristiche esposte più avanti, dimostrano la volontà di scostarsi dallo standard industriale IBM, per creare un apparecchio veramente all'avanguardia. Nell'IBM e compatibili vengono usati i "vecchi" floppy da 5 1/4", molto più delicati, che offrono una minore capacità in Kbyte rispetto ai dischetti da 3 1/2 pollici.

Ovviamente questa scelta ha influito sulla compatibilità software tra la precedente serie di computer prodotti dalla Research Machines, con drive da 5 1/4 pollici, e il Nimbus. L'ostacolo è stato aggirato dotando il Nimbus della possibilità di drive aggiuntivi, compresi quelli da 5 1/4". Ciò significa che il Nimbus è in grado di leggere dischetti nei formati PC-DOS, MS-DOS e CP/M: un enorme vantaggio per chi desideri passare ad un apparecchio dalle prestazioni superiori, senza dover perdere il software precedentemente acquistato.

Sotto i due drive sono collocati due connettori per cartucce ROM, compatibili con quelle disponibili sul 480Z, sebbene la quantità di software su cartucce sia relativamente scarsa. Più a sinistra si trova un connettore che permette l'innesto di un "dongle", ossia di un dispositivo senza il quale è impossibile effettuare copie di programmi (una soluzione al problema della "pirateria"?).

Il software in dotazione al Nimbus comprende il sistema operativo MS-DOS, il *Wordplan* e il *Multiplan* della Microsoft e, infine, le versioni LOGO e BASIC prodotte dalla stessa RM. L'uso di questi programmi, e specialmente della grafica in BASIC, evidenzia immediatamente i notevoli pregi di questo computer.

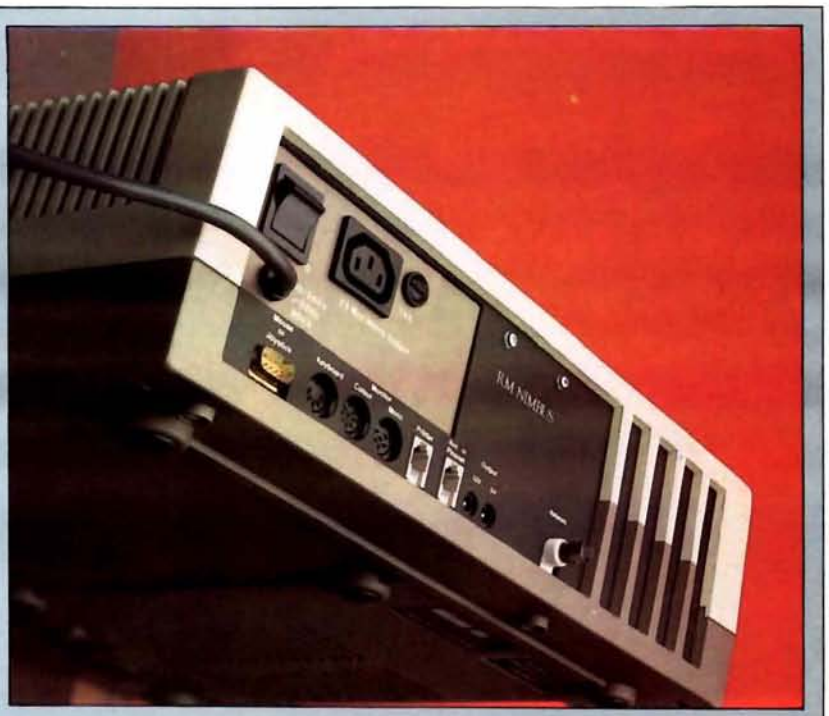
Il BASIC della RM

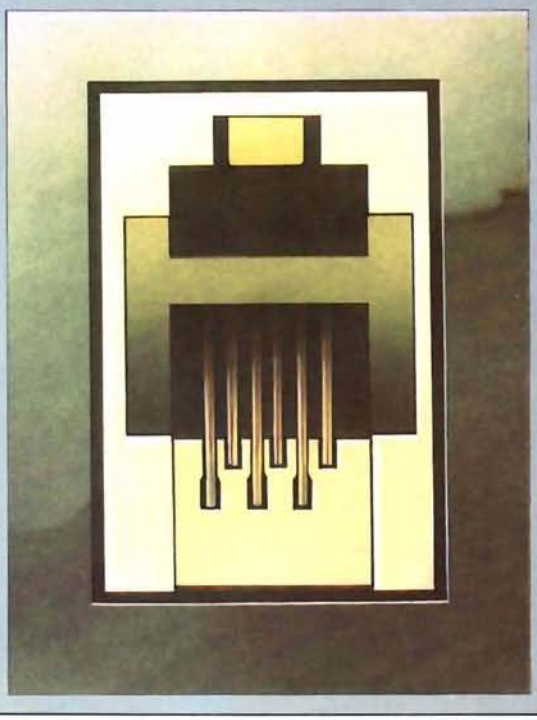
Il BASIC fornito con il Nimbus è uno sviluppo del BASIC RM versione 5. Essendo la Research Machines orientata soprattutto verso il settore didattico, non c'è da meravigliarsi se il BASIC contiene un congruo numero di comandi che facilitano la programmazione strutturata, tra i quali: REPEAT... UNTIL, PROC... ENDPROC e GLOBAL (manca però la DO.. WHILE).

Inoltre, sono stati aggiunti utili comandi per chi desideri realizzare giochi: i comandi JOYX, JOYY, MOUSE e BUTTON, ad esempio, semplificano l'impiego di joystick e mouse.

La grafica e gli effetti audio vengono selezionati in BASIC tramite il comando SET, seguito da opportuni parametri. Per mezzo di SET si possono selezionare i colori dello sfondo e del primo piano (definiti mediante i comandi PAPER, BRUSH e PEN), tracciare cerchi e curve, scegliere la forma dei caratteri, le direzioni e le distanze. Infine, consente anche di selezionare particolari configurazioni predefinite (pattern) utili per arricchire schermi grafici e testuali.

A parte i vari effetti speciali ottenibili sul Nimbus, ciò che colpisce è l'incredibile veloci-



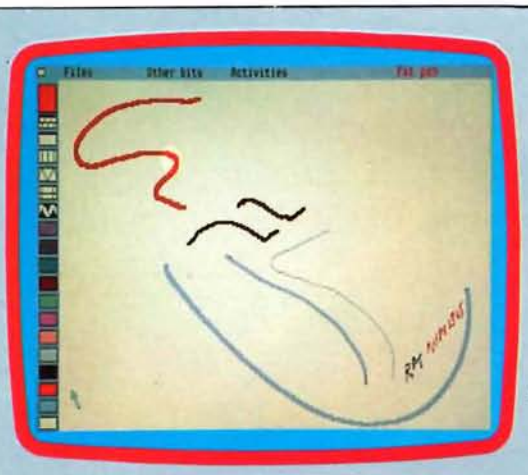


SPAZIO PER ESPANSIONI

Il Nimbus è particolarmente dotato di interfacce: due prese per monitor, una porta per joystick/mouse, un'interfaccia Piconet, una porta per stampante e ben tre uscite di alimentazione per apparecchiature esterne. L'interfaccia Piconet e quella per la stampante adoperano entrambe un insolito connettore di tipo telefonico, al quale le periferiche non possono essere collegate salvo l'acquisto di appositi spinotti.

La presa di tipo telefonico (illustrata qui a fianco) consente lo scambio di segnali seriali, secondo lo standard RS422. I sei contatti disponibili, comunque, consentono numerose combinazioni per altre applicazioni

KEVIN JONES





Il MacNimbus

Tra i programmi in dotazione al Nimbus c'è un package dimostrativo sull'uso della grafica con il mouse. Come tanti programmi del genere, la somiglianza con il MacPaint è piuttosto evidente. Il programma per il Nimbus consente la visualizzazione simultaneamente fino a 16 colori diversi sullo schermo

IAN McKINNELL

tà di esecuzione dei programmi BASIC, paragonabile a quella di programmi in codice macchina su altri apparecchi a 16 bit.

La directory, come di consueto sotto MS-DOS, è suddivisa in sotto-directory. Per passare dall'una all'altra si usa il comando CHDIR (da CHange DIRectory) o più brevemente CD, seguito dal nome del "percorso", ad esempio: CD/BASIC/GIOCHI/GINO oppure CD/BASIC/CONTAB). Il sistema operativo, naturalmente, consente di creare nuove sotto-directory o di eliminare quelle non più utilizzate.

Anche nella dotazione delle interfacce, il Nimbus presenta differenze notevoli rispetto allo standard. Innanzitutto, sul pannello posteriore, è disponibile una presa di corrente tipo A2 per alimentare eventuali periferiche direttamente dal computer. Altre due uscite forniscono 12 volt e 5 volt (utilissime in laboratori didattici, ma anche altrove).

È presente una porta Piconet, sotto forma di presa telefonica secondo lo standard americano, che consente il controllo di un massimo di 30 periferiche, ognuna collegata tramite un modulo esterno, ma tutte pilotate da un'unica porta. L'interfacciamento è di tipo seriale ad alta velocità e corrisponde allo standard RS422.

Chip di I/O
La gestione delle funzioni di ingresso/uscita è affidata a questo circuito integrato

Chip Z80A
Per consentire la lettura e l'elaborazione di dischetto nel formato usato dal 380Z e 480Z, la Research Machines ha incluso questo famosissimo chip come processore secondario

Scheda RAM
Questa particolare versione è dotata di una scheda per l'espansione della memoria

Chip di RAM
La versione standard del Nimbus è dotata di 128 Kbyte di RAM

Connettore per "Soft Key"
Conosciuto anche sotto il nome di porta "dongle", serve per l'innesto di uno speciale dispositivo senza il quale viene inibita la copia non autorizzata del software

Chip a
Cons
gnali s

Unità di controllo per dischetti
Il funzionamento delle unità a dischetti è controllato da questa scheda alloggiata in uno dei connettori di espansione

Unità a dischetti
Le unità a dischetti (1 o 2 a seconda del modello) sono nel formato da 3 1/2 pollici Sony



Sfortunatamente tale standard non è compatibile con il più noto (ma più datato) RS232, cosicché occorre procurarsi gli appositi connettori.
Infine, esiste la possibilità di collegare a rete più computer RM, tramite un cavo coassiale.

Interfaccia per stampante

L'interfaccia per stampante del Nimbus è piuttosto curiosa: oltre ad essere soltanto di tipo seriale, è montata sotto forma di connettore telefonico, identico a quello per la Piconet, con standard RS422.
I connettori per joystick o per mouse, invece, sono i più familiari tipo D a nove poli. Il mouse consente di lavorare in applicazioni simili a quelle del Macintosh della Apple (in effetti il programma dimostrativo per il mouse assomiglia in modo impressionante a quello del Macintosh, anche se privo di alcuni comandi rispetto al MacPaint).
Le prese per monitor sono due, una delle quali è per segnali RGB (in particolare per il monitor appositamente realizzato dalla Cub per la RM). L'altra uscita è per video composito, generalmente adoperato per pilotare i normali monitor a fosfori verdi. Infine, c'è la presa per la tastiera: una normale DIN.
Sul lato posteriore sinistro ci sono quattro connettori per schede di espansione (memoria RAM aggiuntiva, interfacce analogico/digitali, ecc.). Diversamente dalla tendenza generale dei produttori, che quasi obbligano l'utente a farsi montare le schede aggiuntive soltanto da personale autorizzato, la RM fornisce ampie informazioni tecniche su come collegare schede aggiuntive realizzate anche dallo stesso utente.

Una documentazione completa

I manuali in dotazione al Nimbus sono di ottimo livello: contengono informazioni orientate in modo particolare ai principianti sia sul sistema operativo MS-DOS, sia sui dischetti, sia sul sistema in generale. L'unica omissione, in effetti, riguarda la parte dedicata al chip 80186, sebbene ciò interessi realmente soltanto ai programmatori in codice macchina.
Tutto sommato, dunque, il Nimbus è un microcomputer che si merita un posto d'onore, specialmente per le azzardate soluzioni progettuali, così devianti dalla comune tendenza industriale.
Ad un prezzo poco più alto di quello di un IBM PC/AT, il Nimbus offre una velocità di esecuzione nettamente superiore e un BASIC altamente flessibile.
Molto probabilmente, questo apparecchio verrà in gran parte utilizzato in scuole e università, considerate le sue caratteristiche orientate alla didattica. Tuttavia, può darsi che il Nimbus serva a far conoscere il nome della Research Machines Ltd anche molto al di fuori di questo settore così specialistico.

CHRIS STEVENS

NIMBUS RM

DIMENSIONI

mm. 400 x 350 x 110

CPU

80186 della Intel, funzionante a 8 MHz

MEMORIA

128 Kbyte di RAM (espandibile fino a 1 Megabyte)

VIDEO

Modo testuale: 25 x 80 caratteri
Modo grafico: 640 x 250 pixel in quattro colori, 320 x 125 in otto colori

INTERFACCE

Presa per collegamento in rete, porta Piconet, interfaccia per stampante, uscite a 5 volt, 12 volt e tensione di rete (2 Ampère) per alimentare apparecchi esterni, uscite per video composito e RGB

LINGUAGGI DISPONIBILI

BASIC RM, LOGO RM, e PASCAL RM

TASTIERA

Tipo QWERTY, con 83 tasti inclusi i 10 per funzioni e la tastierina numerica

DOCUMENTAZIONE

I manuali (in inglese) sono di ottimo livello e descrivono la quasi totalità delle informazioni sull'intero sistema

PRO

Specialmente nel suo paese di origine (GB) il Nimbus offre un ottimo rapporto tra costo/prestazioni, decisamente migliore di tanti altri apparecchi sul mercato. Il chip scelto come CPU, ossia l'Intel 80186, fornisce una velocità di elaborazione nettamente superiore alla concorrenza

CONTRO

Trattandosi di un microcomputer proiettato verso il futuro, alcune delle interfacce non corrispondono agli standard correnti

Flussi sotto controllo

Lo scambio di dati attraverso i canali dello Spectrum

Come già accennato a pag. 1797, quando si accende lo Spectrum (o si esegue il "reset") il processo di inizializzazione assegna particolari canali ad un gruppo di flussi. La seguente tabella mostra tali abbinamenti:

Canale	Flusso	Descrizione del Canale
K	0	Tastiera e parte bassa dello schermo
K	1	
S	2	Parte principale dello schermo
P	3	Stampante ZX

Comunque sono disponibili altri flussi, nella versione base dello Spectrum, che rimangono normalmente inutilizzati. Si noti che la seguente descrizione si riferisce soltanto a Spectrum sui quali non è montata l'Interface 1, poiché essa (e l'allacciamento a reti locali o di unità Microdrive) introduce cambiamenti sostanziali, che vengono esaminati più avanti.
I canali da 4 a 15, dunque, possono essere liberamente associati a qualsiasi canale di I/O si desideri interfacciare e il sistema operativo conserva i vari abbinamenti in una tabella composta da 38 byte, chiamata STRMS e situata all'indirizzo 23568 (nell'area delle variabili di sistema):



La tabella è così organizzata: i primi sei byte iniziali della tabella sono adoperati dal SO dello Spectrum, mentre i successivi 32 byte corrispondono ai 16 flussi disponibili (a ogni flusso è associato un indirizzo composto da due byte).
Il seguente ciclo in BASIC consente di esaminare il contenuto della tabella appena descritta:

```
10 FOR I = 23568 TO 23604 STEP 2
20 PRINT PEEK I + 256 * PEEK (I + 1)
30 NEXT I
```

Questo visualizza tutti gli elementi a due byte della tabella: i primi tre sono quelli riservati al sistema, gli altri rappresentano i puntatori dei 16 canali, corrispondenti ad altrettanti valori mantenuti in un'altra tabella. L'indirizzo iniziale di questa seconda tabella è conservato nella variabile di sistema CHANS, nella locazione 23631. Per visualizzare l'indirizzo contenuto in CHANS si può usare:

```
PRINT PEEK 23631 + 256 * PEEK 23632
```

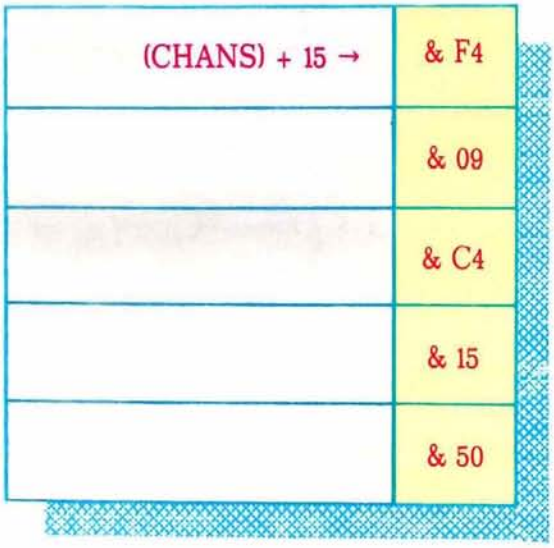
Su questa variabile di sistema non vanno mai eseguite POKE, poiché modificare l'indirizzo iniziale della tabella dei dati sui canali può provocare risultati imprevedibili. Il valore estratto dalla tabella STRMS serve da indice per questa seconda tabella, fornendo lo (scarto + 1) rispetto all'indirizzo iniziale della tabella per un dato canale. Pertanto, l'elemento in STRMS corrispondente ad esempio, al flusso 3 (la stampante), vale 16. Sottraendo 1 si ottiene un offset o scarto di 15: i dati concernenti il canale P si trovano quindi all'indirizzo:

```
(PEEK 23631 + 256 * PEEK 23632) + 15
```

Naturalmente, ciò è vero soltanto se non si sono alterati gli abbinamenti da quando si è acceso il computer.
Se il dato contenuto nella tabella STRMS vale zero, significa che il corrispondente flusso non è stato abbinato ad un canale. Su di uno Spectrum privo di espansioni (e subito dopo l'inizializzazione) valgono zero tutti gli elementi riferiti ai flussi da 4 a 15.
Ogni elemento nella tabella dei canali (chiamata anche CRT, da "Channel Record Table"), ha il seguente formato:

byte-basso	Indirizzo della routine di byte-alto emissione per il canale
byte-alto	
byte-basso	Indirizzo della routine di byte-alto immissione per il canale
byte-alto	
n	Byte che contiene il codice letterale del canale in ASCII (ossia: K, S o P)

Ogni elemento nella tabella STRMS punta al secondo byte di ciascun elemento della tabella in esame, pertanto il puntatore va decrementato di 1 prima di essere adoperato. La tabella CRT (Channel Record Table) è di fondamentale importanza per tutte le operazioni di I/O sullo Spectrum.
Qualora non sia possibile effettuare un'operazione di I/O su di un dato canale, allora il corrispondente elemento di CRT fornisce l'indirizzo di una routine ROM, che visualizza il messaggio Invalid I/O device.
È il caso, ad esempio, del canale S, utilizzabile soltanto in uscita: l'indirizzo che dovrebbe corrispondere alla relativa routine d'immissione è in effetti quello della routine d'errore in ROM.
Esaminiamo l'elemento della tabella per il canale P. Si rammenti che in condizioni normali questo canale è abilitato per la sola emissione (diretta alla stampante ZX). I dati per questo canale, in CRT, sono:



I dati per il canale S iniziano da (CHANS) + 5 e quelli per il canale K da (CHANS)+0, dove (CHANS) è il contenuto nella variabile di sistema CHANS.
Usando i comandi OPEN# e CLOSE# si possono associare flussi inizialmente non utilizzati a uno qualsiasi dei canali appena citati. Ma il vero scopo di questi due comandi è quello di abbinare flussi con i canali in un sistema espanso tramite, ad esempio, l'Interface 1. Comunque, è utile esaminare i due comandi anche in un ambiente non espanso. Ad esempio:

```
OPEN #4;"S"
```

associa il flusso 4 al canale S. Se, dopo aver impartito il comando, si esamina la tabella STRMS, si trova che è variato l'elemento corrispondente al flusso 4.
Nel comando si possono anche adoperare variabili, come in:

```
10 LET n=4
20 LET c$="S"
30 OPEN #n,c$
```

Il comando OPEN ha un effetto identico a quello della OPEN vista in precedenza. Effettuata l'apertura del canale, i comandi PRINT# e LIST# consentono di eseguirvi due tipi di emissione:

```
40 PRINT #4;"Ciao!"
```

visualizza la scritta Ciao! su schermo. LIST #4, invece, visualizza sullo schermo il listato del programma attualmente in memoria. Se il flusso viene abbinato ad un canale che consente anche l'immissione, allora si può adoperare il comando INPUT#.
Comunque, questi comandi speciali possono risultare profiqui soltanto se è presente l'Interface 1, come viene spiegato meglio in seguito.
Terminate le operazioni di I/O su di un particolare canale, esso va "chiuso": vale a dire che va sciolta l'associazione tra il flusso e il canale e ciò si ottiene con il comando CLOSE#.
Nel nostro esempio il canale va chiuso con il comando CLOSE #4.
Qualsiasi ulteriore tentativo di emissione o immissione su di un canale "chiuso" provoca un messaggio d'errore.
I comandi PRINT#, LIST#, INPUT# e INKEY$# possono essere impiegati anche sui normali flussi "privilegiati" da 0 a 3, che il sistema operativo considera sempre "aperti". Pertanto è consentito scrivere:

```
10 CLOSE #2:REM dovrebbe chiudere il canale S
20 PRINT #2;"Ciao!"
```

senza che venga generato alcun messaggio d'errore (il SO provvede a riaprire il flusso prima dell'operazione di emissione). Adesso consideriamo il comando LIST#.
LIST #3, in normali circostanze, equivale a LLIST (il listato del programma viene emesso sul flusso 3 abbinato, come si è visto, al canale P). LIST #0 e PRINT#0 sono comandi piuttosto interessanti, poiché essi servono per visualizzare nella parte inferiore dello schermo (alla quale le normali PRINT non hanno accesso). Si noti, però, che quanto visualizzato in questa zona viene sovrascritto dai prompt e dai messaggi d'errore emessi dal sistema operativo. Sempre in circostanze normali, PRINT#3 equivale a LPRINT.
Per concludere, va detto che è possibile modificare il comportamento di un canale dello Spectrum, facendolo funzionare in modo diverso rispetto alla normalità. Il canale P, ad esempio, è spesso modificato per poter pilotare una stampante Centronics tramite un'apposita interfaccia. Modificando l'indirizzo della routine di emissione nella tabella CRT, i comandi LPRINT e LLIST vengono fatti funzionare tramite una speciale routine in grado di gestire la stampante Centronics. Lo stesso principio, comunque, vale anche per gli altri dispositivi.

PROGRAMMA PER PRINT IN ALTA RISOLUZIONE

Il seguente programma "Hi-Res" intercetta i dati inviati sul canale P dello Spectrum e li passa ad una routine che mette a disposizione un sistema di visualizzazione ad alta risoluzione. Il listato in Assembly può essere convertito in codice usando un qualsiasi programma assembler ("CHAMP", "DevPac" o "Picturesque", ad esempio). In alternativa, si può immettere il codice esadecimale (colonna a sinistra) usando un monitor per codice macchina o il programma caricatore di codice Hex in BASIC, listato qui sotto. Tale programma parte chiedendo l'indirizzo dal quale iniziare a depositare il codice hex (nel nostro caso 65068), dopodiché vanno immessi, nella corretta sequenza, i valori esadecimali (due cifre per volta). Per esempio, la linea di codice:

21 30 FE

va immessa nel seguente modo:

21 <ENTER>
30 <ENTER>
FE <ENTER>

Per terminare l'immissione si digita una S e il programma BASIC passa a chiedere la "checksum" (nel nostro caso 29155), usata per verificare se si sono commessi errori di battitura.
Il nuovo comando creato grazie alla routine Hi-Res ha il formato:

LPRINT AT X,Y;"MESSAGGIO"

dove X deve essere compreso tra 0 e 168 e Y tra 0 e 248. "MESSAGGIO" può essere una qualsiasi variabile numerica o stringa. Si noti che se la stringa da visualizzare col nuovo comando supera il margine dello schermo, la parte eccedente viene visualizzata partendo dall'inizio della stessa riga di schermo. Le coordinate X e Y funzionano in modo identico a quello usato per il comando PLOT.

Programma caricatore di codice Hex

```
10 CLEAR 63999
15 POKE 23658,8
17 DEF FN H(H$)=16*(CODE (H$)-48-7*(H$(1))*9"))
   +(CODE (H$(2))-48-7*(H$(2))*9"))
20 INPUT "Indirizzo iniziale ";a
25 LET I=A
30 INPUT (a), LINE a$
35 IF A$="S" THEN GO TO 90
40 PRINT a,a$: POKE 23692,255
50 FOR I=1 TO LEN A$/2
60 POKE a,FN H(a$(((i*2)-1)TO)): LET a=a+1
70 NEXT i
80 GO TO 30
90 INPUT "CHECKSUM:";D
100 LET C=0: FOR B=I TO A-1: LET C=C+PEEK B: NEXT B
110 IF C=D THEN PRINT "Il codice e' OK!":STOP
120 PRINT "Spiacente, il codice non e' corretto":STOP
```

Listato Assembly

```
                ORG  65068
          PIIAD EQU  22AAH
          UDG   EQU  23675
          CHARS EQU  23606
                ;    L'indirizzo della nuova routine e' posto in CHANS
2A4F5C    ENABL LD   HL,(23631)
010F00          LD   BC,15
09              ADD  HL,BC
01JAFE          LD   BC,DO-IT
71              LD   (HL),C
23              INC  HL
70              LD   (HL),B
C9              RET
                ;    Salva registri e chiama nuova routine di print
E5        DO-IT PUSH HL
C5              PUSH BC
D5              PUSH DE
F5              PUSH AF
CD46FE          CALL DOIT1
F1              POP  AF
D1              POP  DE
C1              POP  BC
E1              POP  HL
C9              RET
                ;    Verifica se a contiene il codice per AT
F5        DOIT1 PUSH AF
3A1DFF          LD   A,(ATFLG)
FE00            CP   0
2008            JR   NZ,GETIP
F1              POP  AF
FE16      ATCHQ CP   22
201D            JR   NZ,CRCHQ
3EFF            LD   A,255
321DFF          LD   (ATFLG),A
C9              RET
FEFE      GETIP CP   254
2809            JR   Z,GETYP
F1              POP  AF
3216FF          LD   (IPOSI),A
211DFF          LD   HL,ATFLG
35              DEC  (HL)
C9              RET
F1        GETYP POP  AF
3217FF          LD   (YPOSI),A
3E00            LD   A,0
321DFF          LD   (ATFLG),A
C9              RET
                ;    Verifica se la stringa e' terminata (codice 13)
FE0D      CRCHQ CP   13
2001            JR   NZ,VCHRQ
C9        SKIPC RET
                ;    Se il codice e' compreso tra 32 e 128 oppure si
                ;    tratta di un UDG cerca i dati in memoria
FE20      VCHRQ CP   32
380C            JR   C,PRNT?
FE80            CP   128
3B1C            JR   C,FCHR
FE90      UDGCQ CP   144
3804            JR   C,PRNT
FEA5            CP   165
3804            JR   C,FUDGC
                ;    Per tutti gli altri codici visualizza un "?"
3E3F      PRNT? LD   A,63
1810            JR   FCHR
                ;    Cerca i dati per UDG o caratteri in memoria
D690      FUDGC SUB  144
210000          LD   HL,0
6F              LD   L,A
29              ADD  HL,HL
29              ADD  HL,HL
29              ADD  HL,HL
EB              EX   DE,HL
2A7B5C          LD   HL,(UDG)
19              ADD  HL,DE
180C            JR   PRNIT
210000    FCHR  LD   HL,0
6F              LD   L,A
29              ADD  HL,HL
29              ADD  HL,HL
29              ADD  HL,HL
EB              EX   DE,HL
2A365C          LD   HL,(CHARS)
19              ADD  HL,DE
010700    PRNIT LD   BC,7
09              ADD  HL,BC
221BFF          LD   (CHRAD),HL
                ;    Verifica la validita' della posizione X,Y
3A16FF          LD   A,(IPOSI)
FEF9            CP   249
D231FF          JP   NC,ERR8
3A17FF          LD   A,(YPOSI)
FEA9            CP   169
3078            JR   NC,ERR5
ED4B16FF        LD   BC,(IPOSI)
CDAA22          CALL PIIAD
321AFF          LD   (PIIPO),A
221BFF          LD   (DFADD),HL
                ;    Estrae primo pixel della sezione da visualizzare
0608            LD   B,8
C5        PRNLP PUSH BC
2A1BFF          LD   HL,(CHRAD)
7E              LD   A,(HL)
2B              DEC  HL
221BFF          LD   (CHRAD),HL
6F              LD   L,A
                ;    Se la sezione di pixel non deve essere spostata
                ;    entro il byte del display allora salta oltre
3A1AFF          LD   A,(PIIPO)
FE00            CP   0
CAE6FE          JP   I,PUTIT
                ;    Sposta opportunamente la sezione nel byte di
                ;    display per ottenere la posizione corretta
47              LD   B,A
2600            LD   H,0
CB3D      ROTLP SRL  L
CB1C            RR   H
A7              AND  A
10F9            DJNZ ROTLP
                ;    Deposita sezione nella giusta posizone di schermo
ED5B1BFF  PUTIT LD   DE,(DFADD)
1A              LD   A,(DE)
AD              XOR  L
12              LD   (DE),A
3A1AFF          LD   A,(PIIPO)
FE00            CP   0
CAFAFE          JP   Z,PST
13              INC  DE
1A              LD   A,(DE)
AC              XOR  H
12              LD   (DE),A
1B              DEC  DE
2A1BFF    PST   LD   HL,(DFADD)
CD1EFF          CALL ULINE
221BFF          LD   (DFADD),HL
C1              POP  BC
10C4            DJNZ PRNLP
                ;    Somma 8 alla posizone X, per passare alla
                ;    successiva posizione di schermo vuota
3A16FF          LD   A,(IPOSI)
C608            ADD  8
                ;    Se la riga di schermo e' terminata, ritorna
                ;    all'inizio senza passare alla riga piu' in basso
FEF9            CP   249
DA12FF          JP   C,STAIT
3E00            LD   A,0
3216FF    STAIT LD   (IPOSI),A
C9              RET
00        XPOSI DEFB 0
00        YPOSI DEFB 0
0000      DFADD DEFW 0
00        PIIPO DEFB 0
0000      CHRAD DEFW 0
00        ATFLG DEFB 0
                ;    Sposta l'indirizzo in HL di un pixel verso l'alto
F5        ULINE PUSH AF
7C              LD   A,H
25              DEC  H
E607            AND  7
200A            JR   NZ,END
7D              LD   A,L
D620            SUB  32
6F              LD   L,A
3804            JR   C,END
7C              LD   A,H
C608            ADD  8
67              LD   H,A
F1        END   POP  AF
C9              RET
                ;    Generazione di errori vari
CF        ERR8  RST  8
0A              DEFB 10
CF        ERR   RST  8
04              DEFB 4
          FINIS END
```

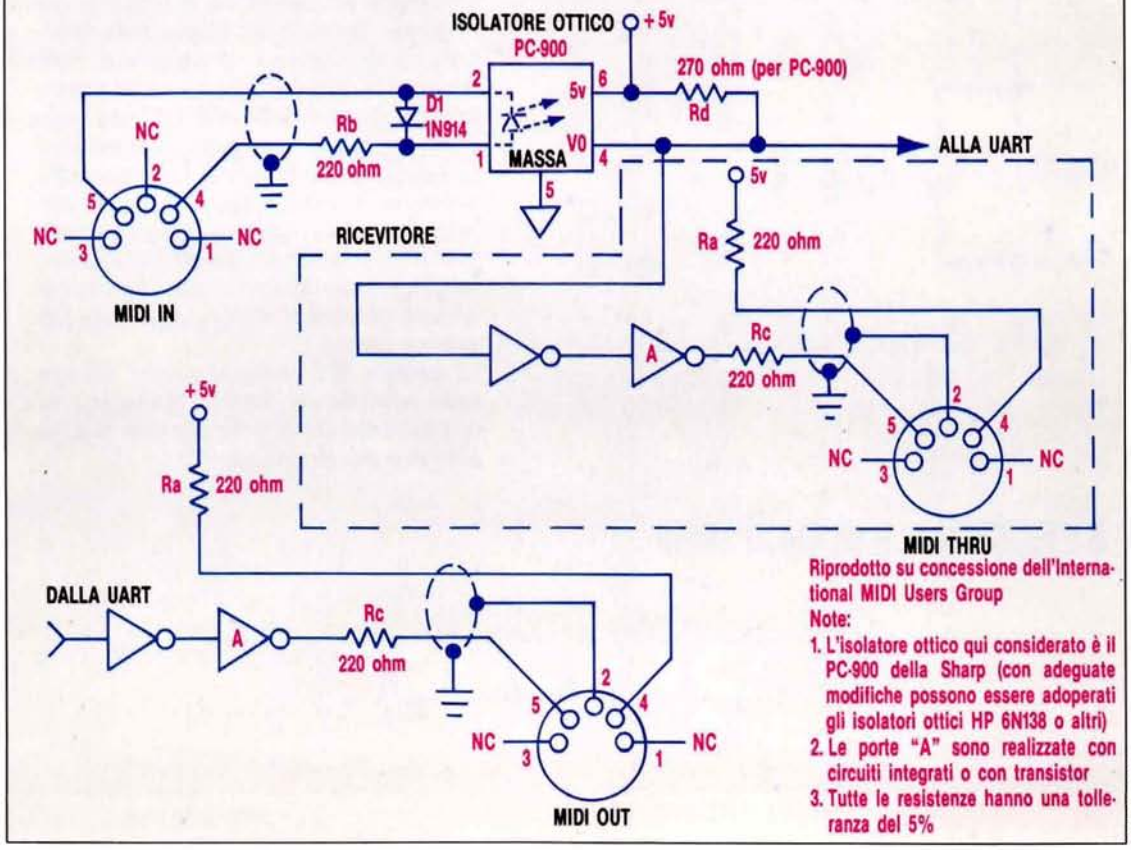
Un'interfaccia MIDI

Un nuovo progetto per far comunicare il Micro BBC e il Commodore 64 con strumenti musicali

Le specifiche della Musical Instrument Digital Interface (MIDI) forniscono indicazioni generali circa l'hardware e il software necessario per creare un sistema nel quale strumenti musicali digitali comunicano tra loro, tramite l'interfaccia MIDI. Poiché le informazioni sono di tipo digitale, non vi sono difficoltà nell'introdurre nel sistema anche un computer, in modo da ottenere un controllo altamente organizzato e coordinato dei vari strumenti. Inoltre, ciò consente di conservare su nastro o su dischetto le sequenze digitali dei codici di controllo. La sezione hardware dell'interfaccia MIDI qui presentata ha un costo veramente accessibile e l'intera realizzazione è di estrema semplicità per chi abbia un minimo di esperienza col saldatore.
Una volta realizzata l'interfaccia, è sufficiente scrivere un programma che consenta di pilotare il sistema usando semplici comandi e parametri (il listato di un simile programma viene fornito in seguito).
Da quando è stato introdotto l'uso dell'interfaccia MIDI, nell'Agosto del 1983, il settore della musica elettronica è piombato nella più grande confusione, poiché a nessuno, all'inizio, era ben chiaro cosa poter fare con l'interfaccia. In parte ciò è dovuto anche al fatto che gli utenti (musicisti e non tecnici elettronici) sono principalmente (e giustamente) interessati ai risultati musicali ottenibili, e non ai particolari tecnici per raggiungere tali risultati.
Durante la descrizione del progetto qui presentato vengono fornite interessanti informazioni tecniche (quelle di cui dispongono i progettisti degli apparecchi elettronici dotati di MIDI): esse sono facilmente comprensibili

L'HARDWARE STANDARD DELLA MIDI

Normalmente l'interfaccia consta di due sezioni: una per la ricezione e una per la trasmissione dei dati, ognuna delle quali ha un proprio connettore, etichettato MIDI IN e MIDI OUT, rispettivamente. Opzionalmente, può esistere un terzo connettore, etichettato MIDI THRU, che si limita ad emettere un'esatta copia dei segnali ricevuti da MIDI IN. Questo è utile per facilitare il collegamento "in serie" di più apparecchi, senza dover aggiungere ulteriori prese MIDI OUT sul trasmettitore. Qui sotto è mostrato il circuito hardware dell'interfaccia MIDI



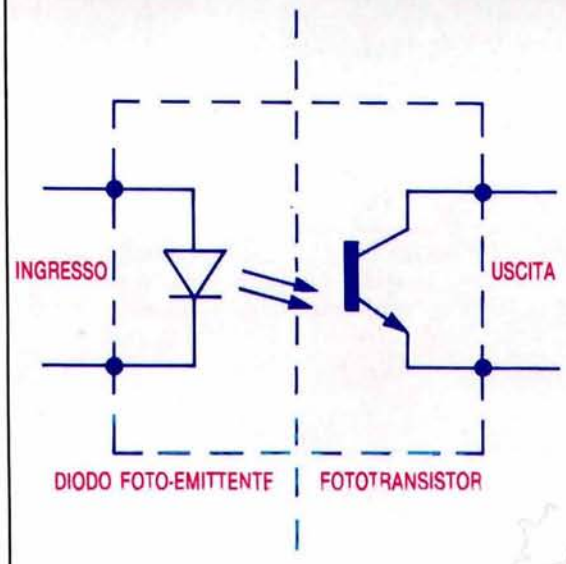
Riprodotto su concessione dell'International MIDI Users Group

Note:
1. L'isolatore ottico qui considerato è il PC-900 della Sharp (con adeguate modifiche possono essere adoperati gli isolatori ottici HP 6N138 o altri)
2. Le porte "A" sono realizzate con circuiti integrati o con transistor
3. Tutte le resistenze hanno una tolleranza del 5%

KEVIN JONES

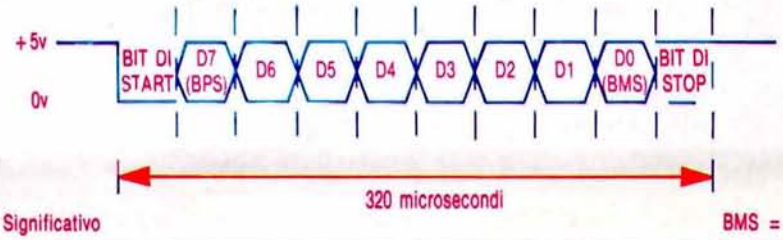
a chi abbia una certa esperienza di programmazione.
Prima di tutto occorre individuare la natura dei dati scambiati tramite la MIDI e ciò è reso più semplice se si considerano i vari strumenti elettronici nel sistema musicale collegati come particolari periferiche di un computer. La distinzione tra alcuni di questi dispositivi può non essere immediatamente evidente: è il caso dei più moderni sintetizzatori che, sebbene formati da due diversi elementi (la tastiera per l'immissione e il generatore di suoni per l'emissione) formano un solo strumento. Il collegamento più ovvio, in tal caso, è quello diretto: premendo un tasto della tastiera si ottiene un suono dal generatore.
Questo sistema è paragonabile a una sofisticata macchina da scrivere elettronica, in cui due componenti distinte (la tastiera e il dispositivo di stampa) possono essere collegate in modo da riprodurre immediatamente quanto digitato sulla tastiera. Tuttavia, il collegamento può avvenire secondo un diverso criterio, ben noto a chi usa abitualmente un word processor: la tastiera è collegabile ad un computer, che elabora lo scritto prima di inviarlo alla stampante.
Nel caso di un sintetizzatore è importante notare che l'interfaccia interna tra la tastiera e il generatore di suoni opera esclusivamente su segnali digitali. In sostanza, la tastiera agisce da puro e semplice controllo, così come in un word processor la tastiera controlla il dispositivo di stampa. Anche in un sintetizzatore musicale, dunque, è possibile inserire un computer tra la tastiera e il secondo componente del sistema (il generatore di suoni).
La musica, in questo caso, non è rappresentata da variazioni di pressione nell'aria (come negli strumenti tradizionali), bensì da una sequenza di "eventi" codificati tramite la tastiera e, in seguito, inviati al sistema di riproduzione. La conversione di tali segnali sonori in effetti, successivamente trasmessi ad un amplificatore audio, spetta ai circuiti del generatore di suoni. Di conseguenza, la MIDI introduce, per la prima volta nella storia della musica, la prospettiva di utilizzare sistemi di controllo svincolati dal concetto di "tastiera musicale".
Un buon esempio (anche se rudimentale) di un conosciutissimo sistema di elaborazione musicale risale a moltissimi anni fa: la pianola. Questo strumento era uno speciale tipo di

CIRCUITO DELL'ISOLATORE OTTICO



Una specifica hardware dell'interfaccia MIDI stabilisce che la linea d'ingresso sia isolata elettricamente dal circuito ricevente, per evitare dannosi ritorni di massa nel sistema, che produrrebbero fastidiosi ronzii alla frequenza di rete. Il problema viene facilmente risolto ricorrendo a un dispositivo chiamato "accoppiatore" o "isolatore" ottico, che fornisce un collegamento non elettrico ma, appunto, ottico tra l'ingresso e l'uscita. Si veda il diagramma a fianco. I due poli dell'ingresso sono collegati a un diodo foto-emittente, simile al LED presente sul pannello della maggior parte delle unità a dischetti. Esso emette luce quando viene applicata una tensione ai suoi capi, con la corretta polarità. Il LED, non visibile dall'esterno, illumina il fototransistor ad esso abbinato, che provvede a regolare il passaggio di corrente ai suoi capi, in base alla quantità di luce ricevuta dal LED: il segnale logico (alto o basso) viene dunque trasferito dall'ingresso all'uscita senza alcuna connessione elettrica.
Le specifiche MIDI stabiliscono inoltre che deve essere sufficiente una corrente di 5mA per accendere il LED e che i tempi di risposta siano di ordine inferiore ai due microsecondi

BYTE SERIALE DI DATI MIDI



BPS = Bit Più Significativo
BMS = Bit Meno Significativo

KEVIN JONES

pianoforte, in grado di suonare brani musicali codificati sotto forma di fori su un rullo di carta. Un meccanismo di lettura, attraverso il quale passava a velocità costante il rullo codificato, provvedeva a convertire il codice in combinazioni di note, agendo sui rispettivi martelletti del pianoforte.
La distanza di ciascun foro dal margine laterale del foglio di carta corrispondeva all'altezza della nota, mentre la distanza in senso longitudinale corrispondeva all'intervallo di tempo tra una nota e l'altra. Ciò consentiva una forma piuttosto elementare di registrazione musicale, alla quale era possibile apportare tutte le correzioni e aggiunte desiderabili: bastava modificare la posizione di uno o più fori nel rotolo di carta. Questo sistema ha notevoli analogie con le schede perforate usate nella programmazione dei primi computer.
Un attuale sistema di elaborazione musicale (il termine "music processor" è ormai di uso corrente) può essere considerato quindi una versione rivista e aggiornata del vecchio metodo per pianola: la memoria del computer (o quella dei dischetti) ha sostituito la carta e un sintetizzatore controllato tramite un'interfaccia ha rimpiazzato il pianoforte. La revisione di un pezzo musicale, oggi, viene fatta direttamente nella memoria del computer, adoperando l'apposito software. Prima di giungere a ciò, comunque, occorre essere a conoscenza di due cose della MIDI: il tipo di trasmissione dei dati impiegato e il formato di tali dati. Questi due elementi corrispondono alle specifiche hardware e software dell'interfaccia. Per adesso consideriamo l'aspetto hardware.

Trasmissione seriale dei dati

Innanzitutto, è necessario sapere come viene inviato un byte da una parte all'altra del sistema. La MIDI funziona come un'interfaccia seriale asincrona, con frequenza di lavoro a 31.25 Kbaud. Ciò significa che può essere inviato un solo bit alla volta e che il dispositivo trasmettitore e quello ricevente non sono sincronizzati tra loro mediante un comune segnale di temporizzazione. Lo svantaggio di una simile scelta è la relativa lentezza, ma in compenso bastano due linee di comunicazione, rendendo il tutto molto economico e facile da realizzare.
Le specifiche MIDI stabiliscono l'impiego di normali connettori DIN a 5 poli (disposti a 180°) o, in alternativa, di connettori professionali XLR a tre poli. Questa scelta è stata fatta intenzionalmente, perché il costo irrisorio di questi componenti ne favorisca la diffusione ad un maggior numero possibile di strumenti.
La critica più frequente rivolta alla MIDI riguarda la sua lentezza che, in sistemi di notevoli dimensioni (come si vedrà in seguito), può anche influire sensibilmente. D'altra parte, è proprio grazie a questa relativa lentezza che è possibile adoperare anche comuni home computer per il controllo di un sistema MIDI.
I dati vengono inviati in gruppi di 10 bit: un byte di dati effettivi, più un bit di start e uno di stop.
Normalmente, in assenza di segnali, la linea si trova a livello logico alto (+5 volt per la MIDI). L'inizio della trasmissione è segnalato da un passaggio al livello basso (0 volt) per la durata di un bit (bit di start).
Quindi vengono trasmessi in sequenza gli otto bit di dati (partendo dal bit più significativo), seguiti da un passaggio a livello alto per la durata di un bit (bit di stop). L'invio di un dato pertanto richiede una durata di 10 bit, ossia 10/31.25 Kbaud = 320 microsecondi. Questo valore è di capitale importanza per progettare un sistema MIDI, come si ha modo di constatare in seguito.
In assenza di segnale, come si è detto, la linea è costantemente a livello alto e il ricevitore verifica a frequentissimi intervalli se c'è stato un cambiamento: un passaggio a livello basso indica l'inizio di una trasmissione. Il ricevitore sa, a questo punto, di dover contare 8 bit di dati e predispone un contatore che attende per un periodo pari a un bit e mezzo. Al termine di questo, il ricevitore è in grado di stabilire se il bit ricevuto è a livello alto oppure basso. I successivi bit vengono letti ad intervalli regolari.
Infine, viene verificata la presenza del bit di stop (livello alto), per controllare la completezza del dato ricevuto. Ricorrendo a questo criterio di trasmissione, non occorre che i segnali di temporizzazione generati rispettivamente dal trasmettitore e dal ricevitore siano perfettamente sincronizzati, il che semplifica l'impiego dell'interfaccia. Ovviamente, le differenze non possono essere di ordine troppo grande: è stata stabilita una tolleranza dell'uno per cento.
L'interfaccia è relativamente veloce, considerando la sua natura seriale (la RS232, ad esempio, ha una velocità massima di 19.2 Kbaud), quindi i cavi di collegamento non dovrebbero superare i 15 metri, per evitare dispersioni di segnale.

Ritardi di trasmissione

A questo punto conviene esaminare la principale limitazione della MIDI, ossia il ritardo di trasmissione tra i vari strumenti. In generale, un messaggio di nota attiva/non attiva consiste in tre byte. Il tempo necessario per l'invio di questi è dunque 3 x 320 microsecondi = 0.96 millisecondi. Adesso si consideri un sistema con un sequenziatore multicanale in cui, ad esempio, occorra inviare 20 messaggi simultanei: l'ultima sequenza viene in effetti inviata con un ritardo di circa 20 millisecondi rispetto alla prima!
In molti sofisticati sistemi i segnali audio prodotti hanno tempi d'innesco sull'ordine dei due-tre millisecondi, pertanto anche un orecchio non particolarmente addestrato può accorgersi di un ritardo di 20 ms.

Il "Nuovo Mondo" II

La seconda parte del listato completo e alcuni suggerimenti per il Micro BBC

Il gioco di simulazione *Nuovo Mondo* è scritto in una versione di BASIC adatta alla maggior parte dei microcomputer e quindi, con qualche piccolo cambiamento, anche al Micro BBC. Le due principali modifiche da apportare riguardano i comandi per ottenere la pulizia dello schermo e per generare una pausa in attesa della pressione di un tasto. Nel primo caso, i possessori di Micro BBC devono sostituire ogni ricorrenza del comando PRINT CHR$(147) con CLS.
Il metodo per ottenere un'attesa di un carattere da tastiera, nella versione Commodore 64 qui presentata, implica la creazione di un ciclo nel quale è impiegata un'istruzione GET. Il ciclo termina quando viene immessa una stringa composta da almeno un carattere non nullo. Il ciclo appare più volte nel programma, sotto la forma generale:

<n. di linea> GET I$:IF I$="" THEN <n. di linea>

Il BASIC del Micro BBC possiede un comando abbastanza simile, ma molto più semplice, col quale è possibile ottenere il medesimo effetto:

<n. di linea> I$ = GET$

Sul Micro BBC, per assegnare la massima quantità di memoria al programma BASIC, conviene adoperare il MODE 7 (quello selezionato automaticamente all'accensione del computer).
Per esser certi che tale modo grafico venga sempre scelto dal programma, basta inserire la seguente linea:

5 MODE 7

```
5300 REM RAPPORTO DI FINE SETTIMANA
5305 PRINT CHR$(147)
5310 S$="        DIARIO DEL CAPITANO*": GOSUB 9100
5312 S$="        -------------------*": GOSUB 9100
5314 GOSUB 9200
5316 PRINT: PRINT"      FINE DELLA SETTIMANA";WK
5318 GOSUB 9200
```

Questo ciclo controlla se durante la settimana ci sono stati decessi fra i membri dell'equipaggio, riducendo, di conseguenza il contatore CN

```
5320 X=0
5325 FOR T=1 TO 16
5330 IF TS(T,2)<>-999 THEN 5350
5335 X=X+1
5340 TS(T,1)=0: TS(T,2)=0
5345 CN=CN -1
5350 NEXT
5355 IF X=0 THEN 5400
5358 PRINT: PRINT
5360 S$="DURANTE LA PASSATA SETTIMANA*": GOSUB 9100
```

Se tutti i membri dell'equipaggio sono morti (ossia CN = 0) allora il gioco è concluso

```
5365 IF CN>0 THEN 5390
5367 S$="E' MORTO TUTTO L'EQUIPAGGIO*": GOSUB 9100
5369 GOSUB 9200: PRINT: GOSUB 9200
5373 S$="INESORABILMENTE LA NAVE VA*": GOSUB 9100
5374 GOSUB 9200
5376 S$=" ALLA DERIVA NELL'OCEANO...*": GOSUB 9100
5378 GOSUB 9200
5380 PRINT: PRINT
5382 S$="      FINE DEL GIOCO*": GOSUB 9100
5384 PRINT: PRINT
5386 GOSUB 9200: END
5388 GOTO 5386
5390 PRINT X;
5392 IF X=1 THEN S$="DECESSO A BORDO*"
5394 IF X<>1 THEN S$="DECESSI A BORDO*"
5396 GOSUB 9100
5398 GOSUB 9200
5400 PRINT: PRINT
```

Se qualche scorta viene esaurita viene visualizzato un messaggio adeguato

```
5405 S$="ADESSO SEI PRIVO DI"
5410 FOR T=1 TO 4
5415 IF PA(T)<>-999 THEN 5440
5420 PRINT S$;P$(T)
5425 PA(T)=0
5428 S$="                    E DI"
5440 NEXT
5450 GOSUB 9200
5455 REM CALCOLA NUOVA DURATA (JL)
```

Il rendimento dell'equipaggio viene calcolato esaminando la matrice TS(,) e, se esso cala di oltre la metà, viene allungata la durata del viaggio

```
5460 X=0
5465 FOR T=1 TO 16
5470 X=X+TS(T,2)
5475 NEXT
5480 IF X>799 THEN 5494
5481 PRINT: PRINT
5482 S$="LA CIURMA E' INDEBOLITA*": GOSUB 9100
5483 S$=" PUO' DARSI CHE IL VIAGGIO*": GOSUB 9100
5484 S$=" RICHIEDA PIU' TEMPO*": GOSUB 9100
5489 EW=EW+((800-X)/800)
5490 REM JL=JL+INT(EW)
5491 JL=8+INT(EW)
5492 PRINT
5494 S$=K$: GOSUB 9100
5496 GET I$: IF I$="" THEN 5496
5499 RETURN
```

Per generare eventi "imprevisti" nella traversata, viene scelto un numero casuale, che è poi adoperato per chiamare una fra più routine (con l'istruzione ON... GOTO)

```
5500 REM GENERATORE DI EVENTI CASUALI
5502 IF RC=RM THEN RETURN
5503 REM ESCE SE TUTTI GLI EVENTI SONO GIA' ACCADUTI
5504 PRINT CHR$(147)
5505 GOSUB 9200
5510 X=INT(RND(1)*RM)+1
5515 REM GENERA UN NUMERO CASUALE TRA 1 E RM
5520 IF RR(X)=1 THEN RETURN
5522 REM ESCE SE EVENTO GIA' USATO DAL PROGRAMMA
5523 RR(X)=1: RC=RC+1
5524 REM MODIFICA L'INDICATORE DI EVENTO E
     AGGIORNA IL CONTATORE DI EVENTI
5525 ON X GOTO 5540, 5570, 5570, 5570, 5570, 5600,
     5700, 5800, 5850, 5900, 5950, 6000, 6050
5528 REM SALTO ALL'APPROPRIATA SEZIONE DI PROGRAMMA
5530 PRINT: S$=K$: GOSUB 9100
5535 GET I$: IF I$="" THEN 5535
5539 RETURN: REM TORNA ALLA ROUTINE PRINCIPALE
```

Alcuni eventi sono funesti...

```
5540 REM EVENTO 1 - UOMO A MARE'
5542 PRINT
5543 S$="DURANTE LA SETTIMANA•": GOSUB 9100
5545 PRINT: GOSUB 9200
5546 S$=" 1 PERSONA E' FINITA IN MARE•": GOSUB 9100
5548 S$=" DURANTE UNA BURRASCA•": GOSUB 9100
5550 PRINT: GOSUB 9200
5552 S$="L'EQUIPAGGIO E' ORA RIDOTTO A•": GOSUB 9100
5554 PRINT CN-1;" MEMBRI"
5558 FOR T=1 TO 16
5559 REM CERCA MEMBRO DA ELIMINARE
5560 IF TS(T,2)=0 OR TS(T,2)=-999
     THEN 5566
5562 TS(T,2)=-999: REM MEMBRO MORTO
5564 T=16
5566 NEXT
5568 GOTO 5530
5570 REM EVENTI 2-5-PERDITA DI SCORTE
5572 X=X-1: REM ORA X PUNTA A SCORTA (1-4)
5574 IF PA(X)=0 OR PA(X)=-999 THEN RETURN
5576 REM NESSUNA AZIONE SE SCORTA GIA' ESAURITA
5578 PRINT
5580 S$="DURANTE LA SETTIMANA•": GOSUB 9100
5582 PRINT: GOSUB 9200
5584 PRINT" UNA PARTE DI";P$(X)
5586 S$=" E' FINITA IN MARE•": GOSUB 9100
5588 PRINT: GOSUB 9200
5590 S$=" NE RESTA PER CIRCA•": GOSUB 9100
5592 PA(X)=PA(X)-INT(PA(X)/(INT(RND(1)•3)+2))
5593 REM RIDUCE SCORTA DI 1/2 1/3 O 1/4
5594 PRINT INT(PA(X)/(CN•PN(X)));
5595 PRINT" SETTIMANE "
5599 GOTO 5530
```

...ma altri sono positivi, come la possibilità di fare una buona pesca, ad esempio

```
5600 REM EVENTO 6 - CATTURA DEL PESCE
5605 X=0
5610 FOR T=1 TO 16
5615 IF TS(T,2)=-999 THEN X=X+1
5620 REM CONTEGGIO DECESSI DELLA SETTIMANA CORRENTE
5625 NEXT
5630 IF CN-X<1 THEN RETURN
5635 REM NON PROSEGUE SE TUTTO L'EQUIPAGGIO E' MORTO
5640 PRINT
5645 S$="DURANTE LA SETTIMANA•": GOSUB 9100
5646 PRINT: GOSUB 9200
5650 S$=" UNO DEGLI UOMINI HA PESCATO•": GOSUB 9100
5655 X=INT (RND(1)•10)+11
5660 REM DA 10 A 20 CHILI
5662 PRINT X;" CHILI DI PESCE"
5665 PRINT: GOSUB 9200
5670 S$="SCORTE DI CARNE•": GOSUB 9100
5675 S$="SUFFICIENTI PER CIRCA•": GOSUB 9100
5678 IF PA(3)=-999 THEN PA(3)=0
5680 PA(3)=PA(3)+X
5685 PRINT INT(PA(3)/(CN•PN(3)));" SETTIMANE"
5690 GOTO 5530
```

Se le scorte di acqua calano, questa routine offre la possibilità di far rifornimento. La linea 5735 genera un numero casuale compreso tra 10 e 20, corrispondente al numero di barili da aggiungere all'elemento relativo alle scorte d'acqua, ossia PA(4)

```
5700 REM EVENTO 7 - TEMPESTA DI PIOGGIA
5705 PRINT
5710 S$="DURANTE LA SETTIMANA•": GOSUB 9100
5715 PRINT: GOSUB 9200
5720 S$=" UNA TEMPESTA PERMETTE•": GOSUB 9100
5725 S$=" UNA RACCOLTA D'ACQUA •": GOSUB 9100
5730 PRINT: GOSUB 9200
5735 X=INT (RND(1)•10)+11
5736 REM DA 10 A 20 BARILI
5740 S$="NUOVE SCORTE D'ACQUA •": GOSUB 9100
5745 S$=" SUFFICIENTI PER CIRCA •": GOSUB 9100
5748 IF PA(4)=-999 THEN PA(4)=0
5750 PA(4)=PA(4)+X
5755 PRINT INT (PA(4)/(CN•PN(4)));" SETTIMANE"
5760 GOTO 5530
5800 REM EVENTO 8 - VENTI FAVOREVOLI
5805 PRINT
5808 S$="VENTI FAVOREVOLI•": GOSUB 9100
5810 S$=" PER TUTTA LA SETTIMANA•": GOSUB 9100
5815 PRINT: GOSUB 9200
5820 S$="LA VELOCITA' AUMENTA E LA•": GOSUB 9100
5825 S$=" DURATA DEL VIAGGIO CALA •": GOSUB 9100
5830 S$=" DI MEZZA SETTIMANA!•": GOSUB 9100
5835 EW=EW-.5
5839 GOTO 5530
```

Il tempo buono influisce positivamente sul rendimento dell'equipaggio. Un fattore scelto casualmente e compreso tra 5 e 15 viene sommato a ciascun elemento della matrice del rendimento TS(,)

```
5850 REM EVENTO 9 - TEMPO BELLO
5855 PRINT
5860 S$="BEL TEMPO PER TUTTA•": GOSUB 9100
5862 S$=" LA SETTIMANA!!•": GOSUB 9100
5865 PRINT: GOSUB 9200
5870 S$="IL MORALE MIGLIORA E GLI•": GOSUB 9100
5875 GOSUB 9200
5880 S$=" UOMINI RECUPERANO FORZE•": GOSUB 9100
5882 FOR T=1 TO 16
5884 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 5888
5886 TS(T,2)=TS(T,2)+INT(RND(1)•11)+5
5888 NEXT
5889 GOTO 5530
```

Queste routine possono causare perdite o danni alle merci trasportate: i fucili possono arrugginire, le stoffe esser divorate dai topi e le bottiglie di medicinali frantumarsi

```
5900 REM EVENTO 10 - PERDITA DEI MEDICINALI
5905 IF OA(1)=0 OR OA(1)=-999 THEN RETURN
5910 PRINT
5915 S$="META' DELLE BOTTIGLIE DI•": GOSUB 9100
5920 S$=" MEDICINALE SI SONO ROTTE•": GOSUB 9100
5925 OA(1)=INT(OA(1)/2)
5930 PRINT: GOSUB 9200
5935 S$="RESTANO SOLTANTO•": GOSUB 9100
5940 PRINT OA(1);" BOTTIGLIE"
5945 GOTO 5530
5950 REM EVENTO 11 - FUCILI ARRUGGINITI
5955 IF OA(2)=0 OR OA(2)=-999 THEN RETURN
5960 PRINT
5965 S$="META' DEI FUCILI SI•": GOSUB 9100
5970 S$=" SONO ARRUGGINITI E•": GOSUB 9100
5972 S$=" NON FUNZIONANO PIU'!!•": GOSUB 9100
5975 OA(2)=INT(OA(2)/2)
5980 PRINT: GOSUB 9200
5985 S$="RESTANO SOLTANTO•": GOSUB 9100
5990 PRINT OA(2);" FUCILI FUNZIONANTI"
5995 GOTO 5530
6000 REM EVENTO 12 - PERDITA DELLE BALLE DI STOFFA
6005 IF OA(4)=0 OR OA(4)=-999 THEN RETURN
6010 PRINT
6015 S$="I TOPI HANNO ROSICCHIATO•": GOSUB 9100
6020 S$=" META' DELLE BALLE DI•": GOSUB 9100
6022 S$=" STOFFA DANNEGGIANDOLE•": GOSUB 9100
6024 S$=" IRREPARABILMENTE•": GOSUB 9100
6025 OA(4)=INT (OA(4)/2)
6030 PRINT: GOSUB 9200
6035 S$="RESTANO SOLTANTO•": GOSUB 9100
6040 PRINT OA(4);" BALLE"
6045 GOTO 5530
6050 REM EVENTO 13 - ALBATRO
6055 PRINT: A$="S"
6060 S$="UN ALBATRO VOLA SULLA NAVE•": GOSUB 9100
6062 GOSUB 9200
6065 S$=" E' UN PRESAGIO FAVOREVOLE•": GOSUB 9100
6068 S$=" CHE SOLLEVA IL MORALE•": GOSUB 9100
6069 S$=" DELL'EQUIPAGGIO•": GOSUB 9100
6070 PRINT: GOSUB 9200
6075 IF PA(3)<(CN•PN(3)•(JL-WK+1)) THEN 6090
6080 REM LA CARNE NON MANCA
6085 GOTO 6122
6090 S$="LE SCORTE DI CARNE SONO•": GOSUB 9100
6092 S$=" QUASI ESAURITE E•": GOSUB 9100
6095 S$=" L'ALBATRO PESA 10 KG!•": GOSUB 9100
6100 PRINT: GOSUB 9200
6105 S$="VUOI ABBATTERLO (S/N)?•": GOSUB 9100
6110 INPUT I$
6112 PRINT: GOSUB 9200
6115 IF LEFT$(I$,1)="S" THEN 6133
6120 S$="FORSE E' UNA COSA SAGGIA!•": GOSUB 9100
6122 PRINT: GOSUB 9200
6125 S$="L'ALBATRO VOLA VIA...•": GOSUB 9100
6130 GOTO 5530
6133 IF OA(2)=0 OR OA(2)=-999 THEN 6180
6135 S$="SPARI UN COLPO..... •": GOSUB 9100
6138 GOSUB 9200: GOSUB 9200
6140 IF RND(1)<.5 THEN 6150
6145 S$=" ...MA SENZA SUCCESSO!•: GOSUB 9100
6148 GOTO 6122
6150 S$="L'ALBATRO CADE SULLA TOLDA•": GOSUB 9100
6155 PRINT: GOSUB 9200
6160 IF PA(3)=-999 THEN PA(3)=0
6162 PA(3)=PA(3)+10: B$="S"
6165 S$=" ADESSO POSSIEDI 10 CHILI•": GOSUB 9100
6167 S$=" DI CARNE... IN PIU'•": GOSUB 9100
6170 S$=" MA DA ORA LA FORTUNA•": GOSUB 9100
6172 S$=" PUO' VOLTARTI LE SPALLE!•": GOSUB 9100
6174 GOTO 5530
6180 S$="NON HAI NEMMENO UN FUCILE!•": GOSUB 9100
6190 GOTO 6122
```

Gli eventi descritti sopra hanno un'influenza relativa sul viaggio: esistono altri sei eventi dall'effetto molto più drastico, selezionati casualmente da questa routine

```
6500 REM EVENTI PRINCIPALI
6508 X=INT(RND(1)•10)+1
6510 ON X GOSUB 6530, 6700, 6800, 6900, 7000, 7050
6520 RETURN
```

DATABASE

L'undicesima tabella della mappa di memoria del Commodore 64. Per concessione della Commodore Business Machines

INDIRIZZO ESADECIMALE	ETICHETTA	LOCAZIONE DECIMALE	DESCRIZIONE
032C-032D	ICLALL	812-813	**KERNAL CLALL Routine Vector** Vettore per routine CLALL del KERNAL
032E-032F	USRCMD	814-815	**User-Defined Vector** Vettore definito da utente
0330-0331	ILOAD	816-817	**KERNAL LOAD Routine Vector** Vettore per routine LOAD del KERNAL
0332-0333	ISAVE	818-819	**KERNAL SAVE Routine Vector** Vettore per routine SAVE del KERNAL
0334-033B		820-827	**Unused** Locazione non usata
033C-03FB	TBUFFER	828-1019	**Tape I/O Buffer** Memoria "buffer" per operazioni di I/O su nastro
03FC-03FF		1020-1023	**Unused** Locazione non usata
0400-07FF	VICSCN	1024-2047	**Sprite Data Pointers** Puntatori ai dati per gli sprite
07F8-07FFF		2048-40959	**Normal BASIC Program Space** Area normalmente riservata ai programmi BASIC
8000-9FFF		32768-40959	**VSP Cartridge ROM- 8192 Bytes** Area riservata alle cartucce ROM (8192 Byte)
A000-BFFF		40960-49151	**BASIC ROM- 8192 Bytes (or 8K RAM)** Area riservata alle ROM del BASIC (circa 8 Kbyte)
C000-CFFF		49152-53247	**RAM-4096 Bytes** Area di RAM - 4096 Byte
D000-DFFF		53248-57343	**Input/Output Devices and Color RAM or Character Generator ROM** Area riservata ai dispositivi di I/O e alla RAM del colore oppure al Generatore di caratteri ROM
E000-FFFF		57344-65535	**KERNAL ROM - 8192 Bytes (or 8K RAM)** ROM del KERNAL - 8192 Byte (oppure 8 Kbyte di RAM)

Il sintetizzatore "Echo"

Un package per accedere alle capacità musicali del Commodore 64 e del Micro BBC

Il pacchetto Echo, della Leasalink Viewdata Ltd (LVL), comprende un'autentica tastiera musicale a tre ottave e mezzo e relativo software di controllo. Viene fornito assieme a un manuale e un cavo d'interfaccia a 20 linee per collegare la tastiera al computer, tramite la porta utente. La versione Echo è corredata da due cavi: quello standard per il Micro BBC e un piccolo cavetto con adattatore per il collegamento al Commodore 64.
L'hardware dell'Echo è abbastanza semplice, poiché la maggior parte del lavoro viene svolta dal software (riducendo così anche i costi di produzione). Inoltre il software per entrambi i computer si basa sulle stesse norme di funzionamento.
L'Echo offre due distinte funzioni di suono: il modo "organo" e quello "sintetizzatore". Il primo consente di selezionare i diversi strumenti disponibili premendo uno dei tasti del computer. Gli strumenti vanno dalla chitarra hawaiana al violoncello e al clavicembalo, ma ciascun computer ha un proprio sistema di selezione. Nella parte alta dello schermo viene visualizzata una serie di parametri che consentono di modificare la tonalità dello strumento suonato.
La versione di software per il Commodore 64 permette di introdurre le funzioni "tremolo" e "vibrato" (ma non contemporaneamente) o di passare da un accordo maggiore ad uno minore e viceversa. Questi effetti sono controllati dai tasti numerici del computer.
Il software in versione BBC utilizza invece i tasti funzione per selezionare il "tremolo" e gli accordi maggiori o minori. Ricorrendo a questi tasti supplementari e alla maggior velocità d'elaborazione del processore, i programmatori hanno introdotto ulteriori funzioni nel software per il Micro BBC. Se si desidera dare maggior ritmo e incisività ai pezzi suonati sulla tastiera Echo, basta premere uno dei tasti funzione per generare i suoni del basso, dei piatti o di altri strumenti ritmici. Sfortunatamente però il sistema non consente di associare un sottofondo ritmico ai brani eseguiti sulla tastiera.

Controllo della tonalità

I controlli del tono vengono visualizzati nella parte inferiore dello schermo. Mentre la versione per il Micro BBC usa i tasti del cursore per innalzare o abbassare il tono di una nota, quella per il Commodore 64 ricorre ai tasti < e >. Il fatto che entrambe le funzioni siano indicate sullo schermo dall'etichetta PITCH MINUS suscita qualche perplessità sulla qualità del software per il Commodore 64. Pur riconoscendo che in generale le voci generate dai tasti "strumentali" somigliano assai poco agli strumenti che vogliono imitare (d'altra parte tale difetto è comune alle applicazioni della musica elettronica, non solo ai programmi per computer), gli effetti di suono dell'Echo appaiono particolarmente scadenti. Il violino, la viola e il violoncello della versione Commodore 64, ad esempio, assomigliano tutti a un organo suonato a differenti tonalità nel tentativo di imitare tali strumenti.
Entrambe le versioni prevedono il modo "sintetizzatore" i cui parametri dipendono dalle funzioni programmate nel chip audio. Come nel modo "organo", la versione Commodore permette di cambiare i parametri d'inviluppo della forma d'onda (premendo gli appositi tasti del computer fino ad ottenere il valore desiderato) e di programmare il filtro. Il modo "sintetizzatore" della versione BBC sfrutta



PAUL CHAVE

Menu musicali
Queste immagini illustrano il funzionamento nel modo "organo" del Commodore 64 e nel modo "sintetizzatore" del Micro BBC. Le due versioni del software, apparentemente simili, presentano in realtà numerose differenze. Entrambe ricorrono ai tasti del computer per modificare i suoni, ma la versione BBC offre una varietà di funzioni di selezione assai superiore a quella del Commodore 64

SINTETIZZATORE ECHO

TASTIERA

37 tasti del tipo a pianoforte, a corsa completa

INTERFACCE

Per il Micro BBC: un semplice cavetto a 20 linee da collegare alla porta utente. Per il Commodore 64 c'è un cavetto supplementare dotato di adattatore

SOFTWARE

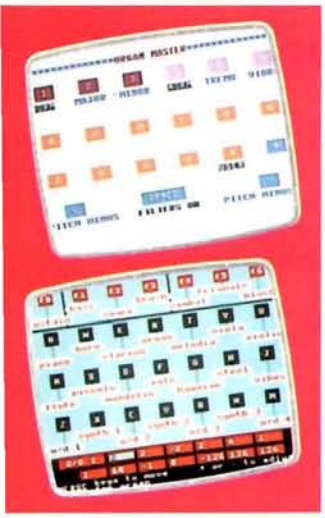
Per il Micro BBC esiste la versione su nastro e quella su dischetto mentre per il Commodore 64 c'è soltanto la versione su cassetta

PRO

Il sistema Echo consente di sfruttare abbastanza bene le capacità audio dei due computer, tramite una tastiera musicale "autentica"

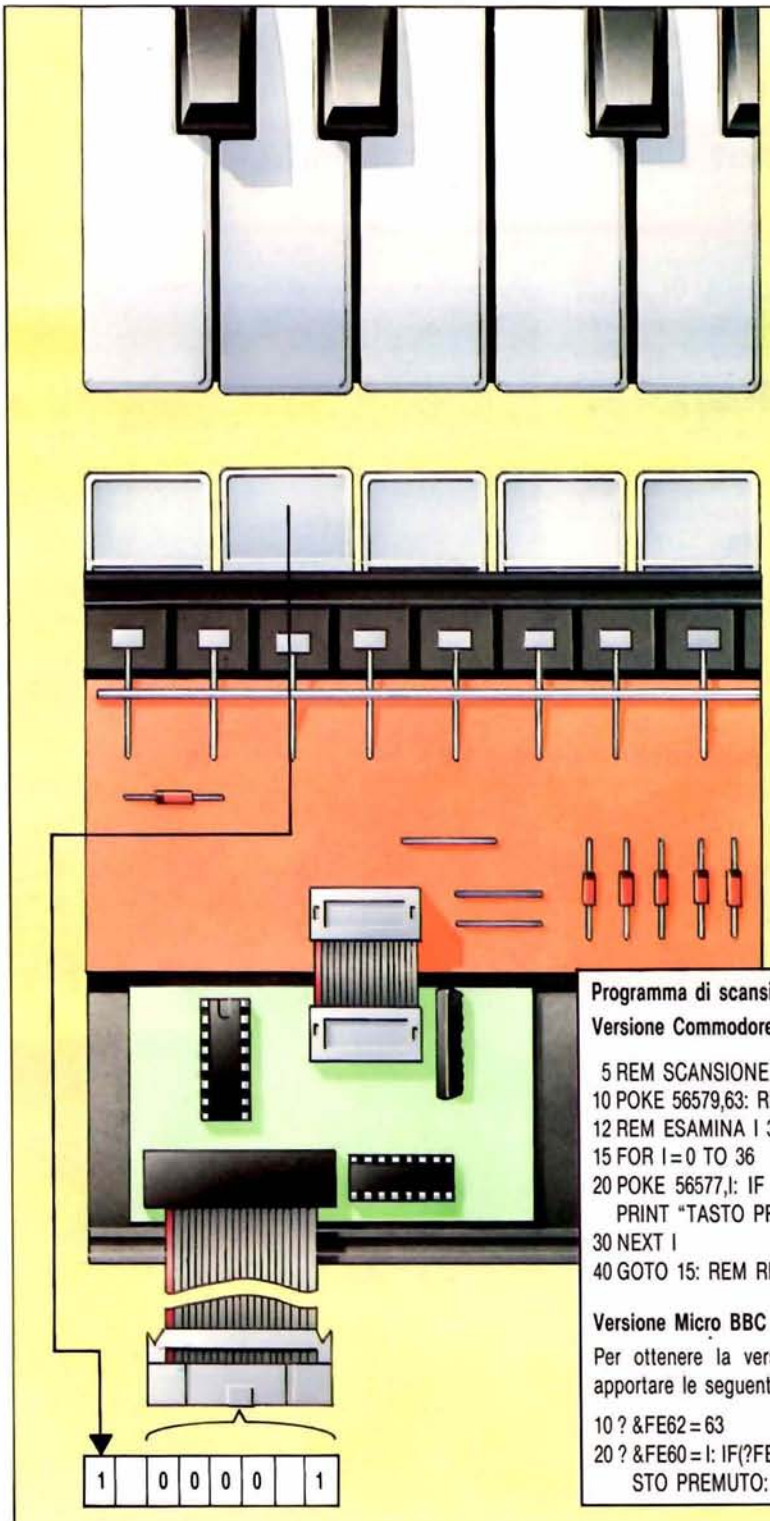
CONTRO

Il software presenta qualche lacuna che, se colmata, migliorerebbe notevolmente la qualità dei suoni emessi



Dalla tastiera al processore

La tastiera Echo della LVL è la prima "vera" tastiera musicale, prodotta per il Micro BBC e il Commodore 64, che accede direttamente ai chip del suono senza la presenza di una costosa interfaccia



LA SCANSIONE DELL'ECHO

La tastiera dell'Echo, collegata alla porta utente del Micro BBC o del Commodore 64, può essere facilmente "esplorata" dal software. I sei bit del registro dei dati della porta utente vengono posti in uscita, per poter inviare informazioni alla tastiera, mentre il bit più significativo (bit 7) viene selezionato per la ricezione dei segnali provenienti da essa. Per individuare quale tasto venga premuto sulla tastiera sono necessarie due operazioni: per prima cosa depositare (con una POKE) un valore compreso tra 0 e 36 nel registro dei dati della porta utente (la tastiera dell'Echo possiede 37 tasti numerati da 0 a 36 a partire da destra): tale valore definisce il tasto sottoposto ad esame; quindi verificare lo stato del bit 7 del registro dei dati (ponendo il suo valore in AND con 128). Se il tasto è premuto il valore del bit 7 è 1, altrimenti vale 0. Quindi, per esaminare l'intera tastiera, va impiegato un ciclo iterativo che verifichi in successione i 37 tasti. Il seguente programma illustra come programmare questa funzione in BASIC.

Programma di scansione dell'Echo

Versione Commodore 64

```
 5 REM SCANSIONE DELL'ECHO SUL CBM64
10 POKE 56579,63: REM BIT 0-5 IN USCITA
12 REM ESAMINA I 37 TASTI DELLA TASTIERA
15 FOR I=0 TO 36
20 POKE 56577,I: IF (PEEK(56577) AND 128)<>0 THEN
   PRINT "TASTO PREMUTO: ";I
30 NEXT I
40 GOTO 15: REM RIPETIZIONE ALL'INFINITO
```

Versione Micro BBC

Per ottenere la versione adatta al Micro BBC basta apportare le seguenti modifiche:

```
10 ? &FE62=63
20 ? &FE60=I: IF(?FE60 AND 128)<>0 THEN PRINT "TA-
   STO PREMUTO: ";I
```

KEVIN JONES

invece il comando ENVELOPE. Quando questo viene richiamato, il sistema chiede all'operatore quale dei quattro sintetizzatori definiti dall'utente (anziché quelli presenti nell'organo) deve modificare. Quindi visualizza i 14 parametri relativi al comando ENVELOPE, che vengono selezionati con i tasti "destra/sinistra" del cursore. I relativi valori si possono modificare ricorrendo ai tasti "alto/basso" del cursore. Eseguite queste operazioni, il sistema è pronto a generare la nota suonata sulla tastiera.

Il software permette al computer di comunicare con la tastiera inserendo valori compresi tra 0 e 36 nei 6 bit meno significativi del registro dati della porta utente. Il package Echo adotta la seguente tecnica di scansione della tastiera: quando vengono premuti più tasti insieme, il sistema registra solo il primo battuto. Purtroppo ciò non consente di usare i tasti in rapida successione perché un tasto battuto deve tornare alla posizione "di riposo", prima che il software possa prendere in esame il successivo.

Circuiti audio

Inizia il progetto hardware dell'interfaccia MIDI per strumenti musicali

Il componente principale dell'interfaccia è un circuito ACIA (Asynchronous Communications Interface Adaptor) che converte i dati paralleli del computer nel formato seriale, richiesto dalla Midi, e viceversa. Di solito l'ACIA è in un unico circuito integrato: nel nostro caso è il chip MC6850, utilizzato su molti home computer per gestire lo scambio asincrono di dati tra le cassette e le interfacce del tipo RS232. Altri componenti del progetto sono un clock d'interfaccia, un accoppiatore ottico e tre prese d'ingresso e di uscita: tutto ciò forma la base dell'interfaccia. Le necessarie connessioni alla porta del computer, sia per il Commodore 64 sia per il Micro BBC, vengono descritte in seguito.
Internamente il circuito ACIA contiene una

ELENCO DEI COMPONENTI

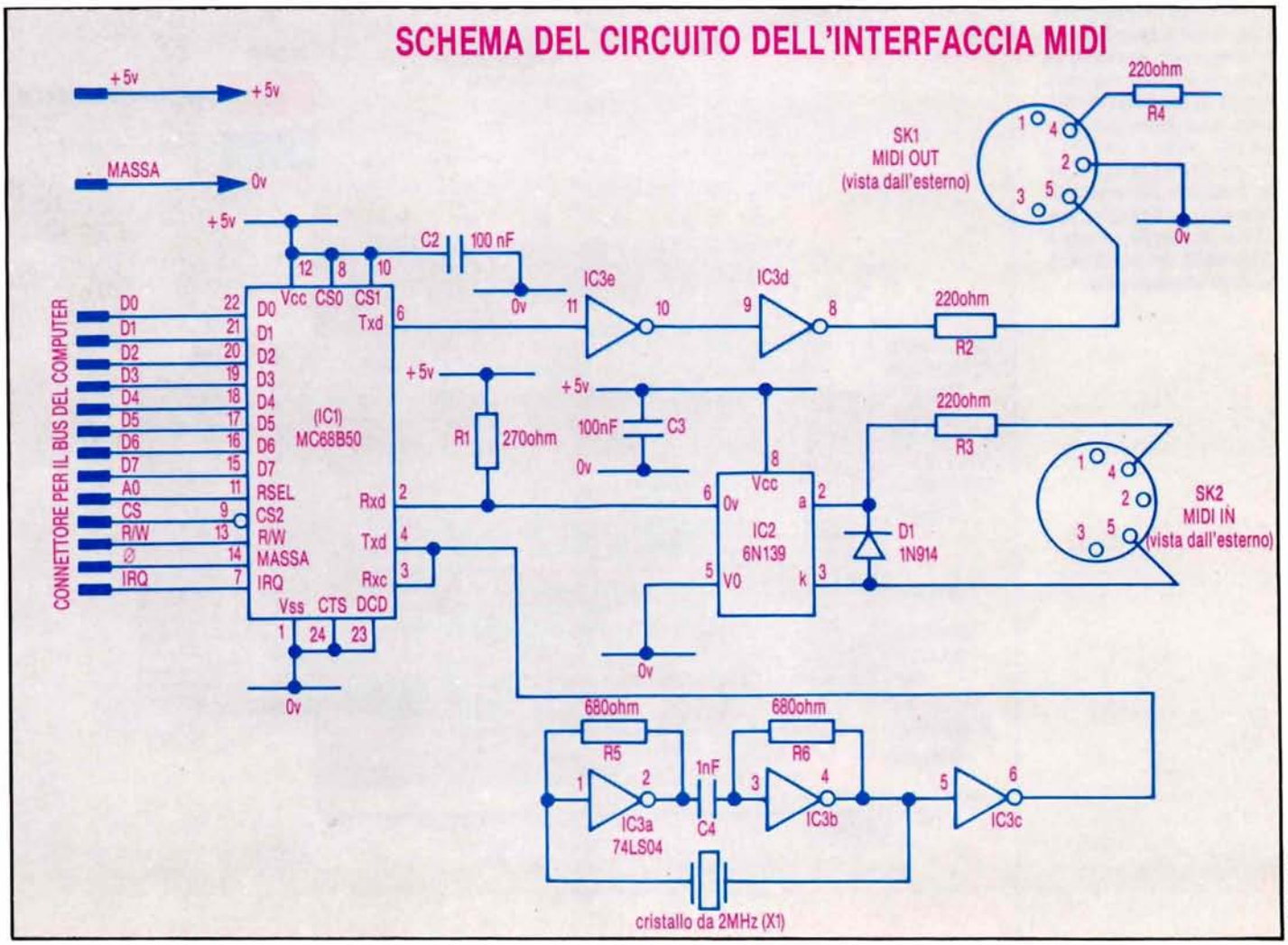
Quant.	Rif.	Descrizione
1	C4	condensatore a policarbonati da 1nF
2	C2/C3	condensatore a policarbonati da 100nF
1	R1	resistenza da 270 ohm
3	R2/R3/R4	resistenza da 220 ohm
2	R5/R6	resistenza da 680 ohm
1		rocchetto di filo elettrico per "Wire wrap"
2	SK1/SK2	prese DIN a 5 piedini (disposti a 180°)
1	IC1	chip ACIA MC68B50
1	IC2	chip optoisolatore 6N139
1	IC3	invertitore esadecimale 74LS04
1		presa DIL a 24 piedini
1		presa DIL a 8 piedini
1		presa DIL a 14 piedini
1	X1	cristallo da 2.0 MHz
1	D1	diodo 1N914

1 scheda con due connettori a pettine per l'assemblaggio dei componenti

Collegamenti del circuito

L'accoppiamento ottico è ottenuto con l'optoisolatore 6N139 (IC2). I segnali di clock per la trasmissione non vengono generati dal clock del computer, ma da un oscillatore a 2 MHz, realizzato con il cristallo e con IC3a e IC3b. IC3c, d ed e forniscono le funzioni di bufferizzazione per il trasmettitore e il generatore di clock. Poiché il chip ACIA MC6850 è compatibile con il bus della CPU 6502, è possibile collegare i micro basati sul 6502 alle linee di Lettura/Scrittura, dell'IRQ e del Clock. Il canale d'ingresso del clock per l'ACIA è stato etichettato con la sigla E, per evitare confusioni con i canali di trasmissione del clock TXD e TXC. Poiché le due linee di selezione del chip, CS0 e CS1 sono collegate alla linea di alimentazione a 5 volt, esse risultano sempre a livello alto. Il solo canale utile per la selezione del chip, è CS2. Tale linea viene posta a livello basso da una serie di indirizzi, per consentire sia l'esecuzione di operazioni di lettura e di scrittura sul bus dei dati sia l'accesso ai registri interni dell'ACIA

SCHEMA DEL CIRCUITO DELL'INTERFACCIA MIDI



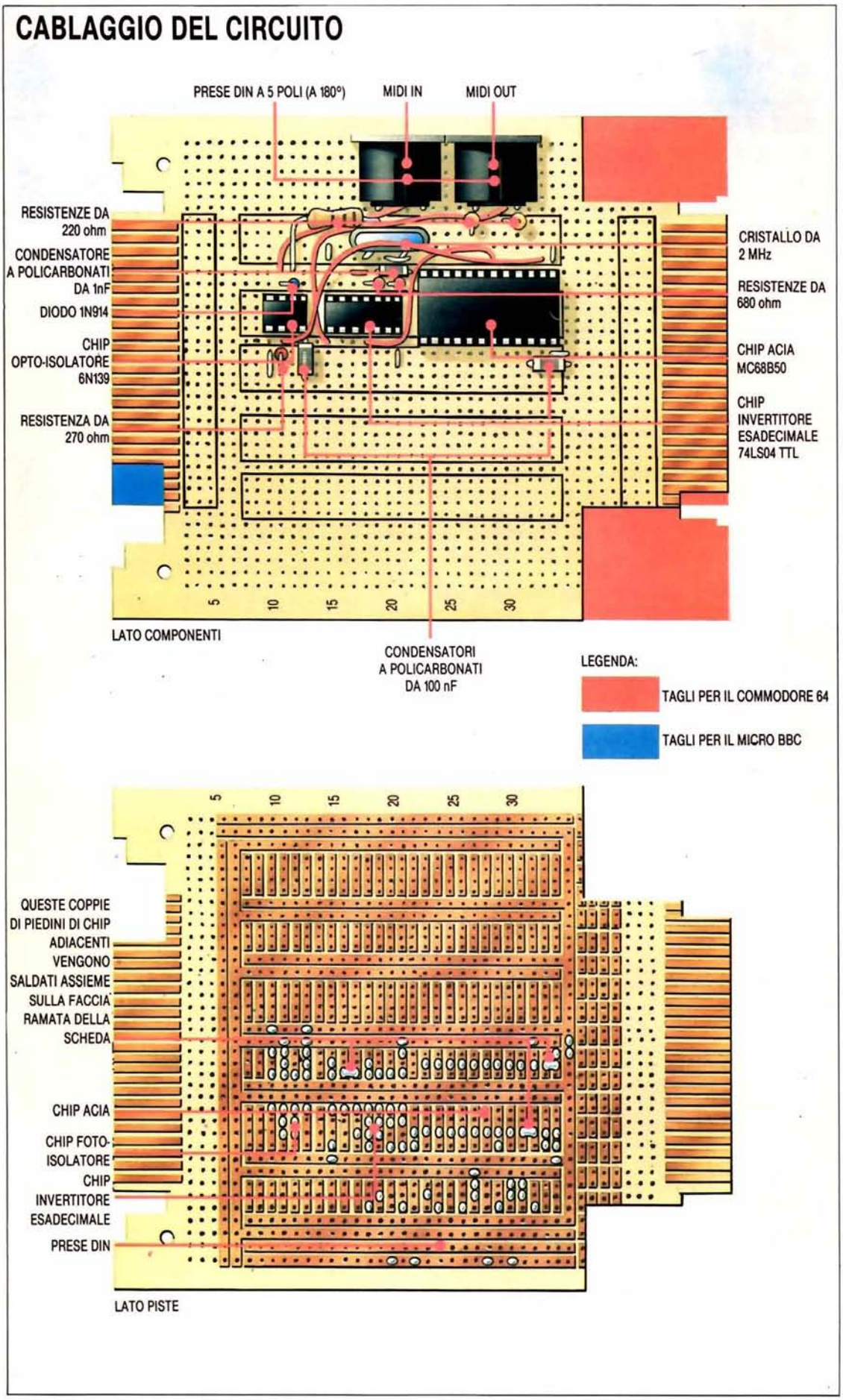
KEVIN JONES

Realizzazione della MIDI

I componenti principali dell'interfaccia MIDI sono montati su una speciale scheda prestampata dotata, alle due estremità, di una coppia di connettori a pettine ramati su entrambe le facce: quello di destra collega la MIDI al Commodore 64, quello di sinistra al Micro BBC. Prima di montare i componenti occorre tagliare e rimuovere alcune parti della scheda. I tagli vanno eseguiti solo sull'estremità della scheda relativa al computer che si vuol collegare all'interfaccia. I ponticelli, realizzati usando un normale filo elettrico per "Wire wrap" (ossia molto sottile), vanno montati sul lato componenti della scheda, saldando le due estremità (previamente spellate) sulle piste ramate sottostanti. Tre ponticelli vengono realizzati sulla faccia delle piste ramate, collegando con lo stagno della saldatura alcuni piedini adiacenti di IC (vedere illustrazione in basso).

Per primi vanno saldati i componenti "passivi", ossia: le resistenze, i condensatori (facendo particolare attenzione a non tralasciare il ponticello situato stto il condensatore C3) e le prese DIL e DIN. Quindi si passa ai cavetti di collegamento, al crisallo da 2MHz e al diodo (con l'estremità marcata da una banda colorata rivolta verso destra, guardando dall'alto). Infine si innestino i chip nei rispettivi zoccoli, facendo attenzione all'orientamento delle tacche. La descrizione conclusiva del progetto riguarda il collegamento dell'interfaccia al computer e il suo collaudo

CABLAGGIO DEL CIRCUITO

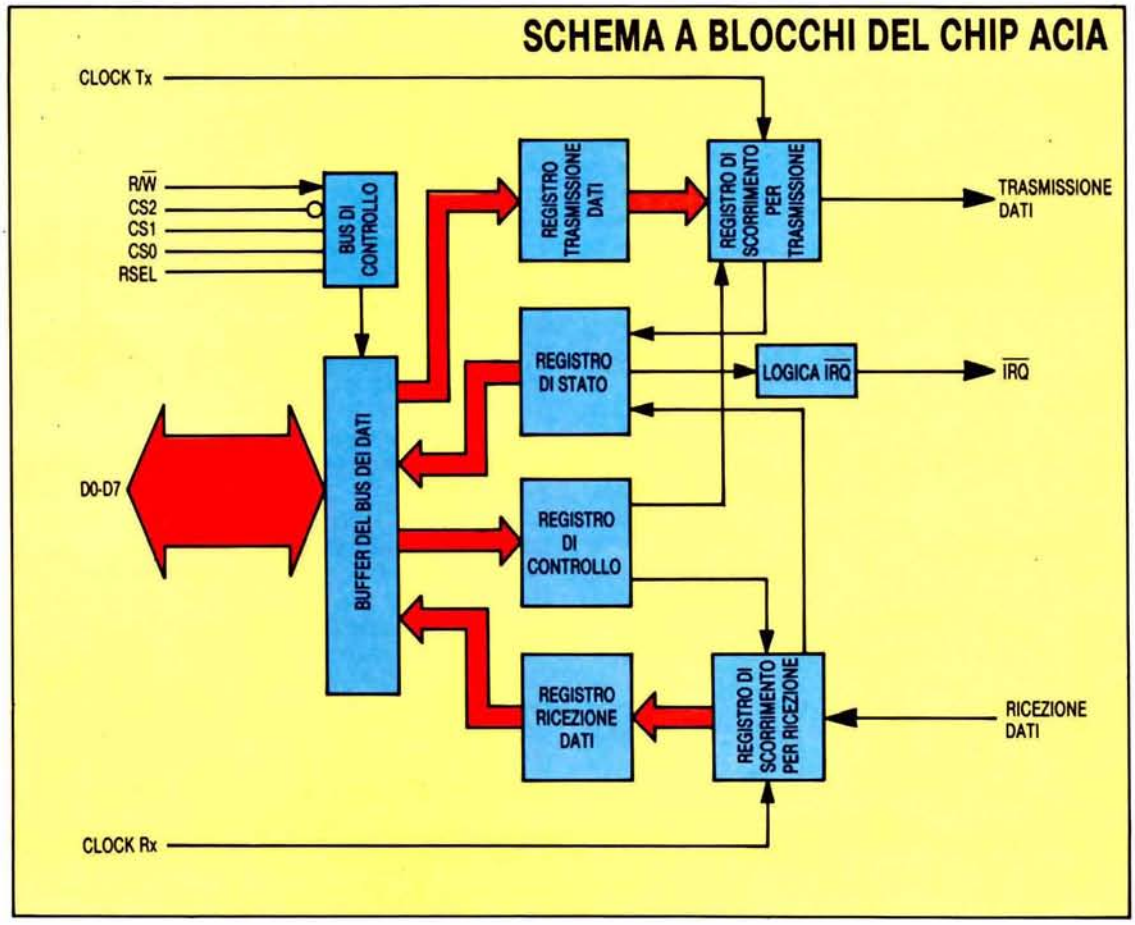


serie di registri a 8 bit. La linea d'ingresso per la selezione del registro (RSEL) permette di accedere ai registri dal bus dei dati. Non è indispensabile comprendere il funzionamento dei registri dell'ACIA per utilizzare il software di controllo fornito in seguito. Tuttavia, per coloro che volessero cimentarsi per conto proprio nella programmazione dell'interfaccia, ecco le informazioni necessarie:

- Il "registro di stato" è composto da 8 bit che descrivono lo stato attuale del circuito ACIA. Si può esaminarne il contenuto con una "lettura" dell'ACIA dopo aver posto la linea RSEL a livello logico 0.
- Il "registro di controllo" contiene 8 bit che controllano il funzionamento dell'ACIA. Per accedere al registro occorre eseguire un'operazione di "scrittura" sull'ACIA con la linea RSEL a livello logico 0.
- Il "registro di scorrimento per la trasmissione" (TSR) esegue la conversione da parallelo a seriale necessaria per trasmettere un byte di dati. Il registro riceve un byte dal "registro di trasmissione dei dati" (TDR) ogni volta che quest'ultimo è completo, purché sia conclusa la trasmissione del byte precedente. Ciò pone a 1 il bit 1 del registro di stato. Il byte viene poi scaricato dal registro alla velocità indicata dal clock di trasmissione ed è in questa fase che ai suoi 8 bit vengono aggiunti i bit di start e di stop. Il TSR è un registro "interno" e non può essere raggiunto direttamente attraverso il bus dei dati.
- Il "registro di trasmissione dei dati" (TDR) è un "buffer" (deposito temporaneo di memoria) interposto tra il TSR e il bus dei dati del sistema. Il byte da trasmettere viene caricato in questo registro eseguendo una "scrittura" sull'ACIA, dopo aver posto al valore logico 1 la linea di selezione del registro (RSEL). Tale operazione azzera il bit 1 del registro di stato. Quindi, per non perdere informazioni, il TDR non deve essere mai caricato se il suo bit di stato è a 0, poiché ciò indica che il byte precedente è ancora all'interno del TDR, in attesa di essere trasmesso.
- Il "registro di scorrimento per la ricezione" (RSR) esegue la conversione da seriale a parallelo per la ricezione dei dati dal sistema MIDI. Dopo aver ricevuto un byte completo, il registro lo trasferisce all'RDR e pone a 1 il bit di stato 0. Se il bit conteneva già il valore, l'RSR seleziona anche il bit 5 per indicare che il byte precedente non era stato letto dalla CPU e che l'informazione è perduta.

Se il byte ricevuto manca dei bit di start e stop (pag. 1815), il registro seleziona il bit di stato 4; ciò può verificarsi quando la linea d'ingresso seriale è disturbata da un "rumore elettrico". Come il TSR, anche l'RSR non può essere raggiunto direttamente dal bus.

- Il "registro di ricezione dei dati" viene caricato ogni volta che l'RSR riceve un byte completo: contiene quindi l'ultimo byte ricevuto. Anche in questo caso, occorre eseguire una "lettura" del chip ACIA, ma con la linea RSEL al valore logico 1, per accedere al registro. L'operazione azzera il bit di stato 0.

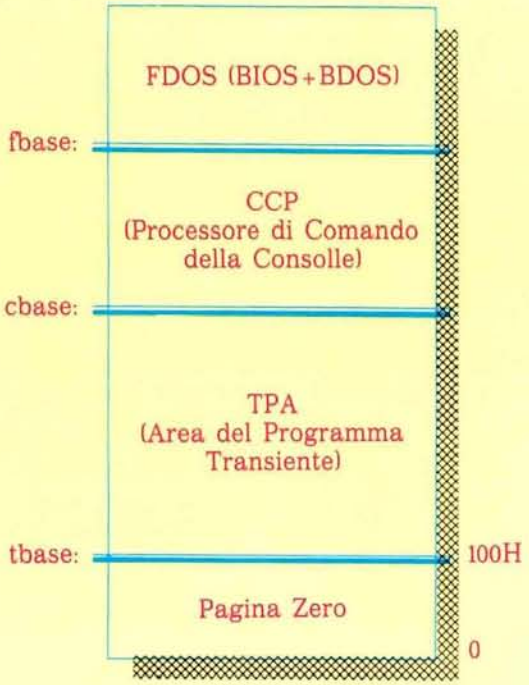


Un futuro promettente

La disposizione dei vari moduli del CP/M nella memoria del computer

Il CP/M in memoria

Il diagramma mostra la disposizione in memoria dei diversi moduli del CP/M. Le variabili di sistema, gli "indirizzi d'ingresso" (entry point) e il programma di caricamento del sistema (bootstrap) si trovano nella parte bassa della memoria, alla pagina zero. Sopra c'è l'Area del Programma Transiente (TPA), il cui limite superiore può essere modificato in base alla memoria disponibile. Comunque sia, i file molto estesi possono tranquillamente soprascrivere il Processore di Comando della Consolle (CCP), poiché viene ricaricato in memoria dopo l'esecuzione dei programmi o dei comandi esterni. La parte alta della memoria accoglie i moduli BIOS e BDOS che partecipano all'esecuzione dei comandi e non devono essere soprascritti



I tre moduli principali del CP/M (CCP, BIOS e BDOS) vengono depositati nella parte superiore della memoria e i loro indirizzi iniziali dipendono dalla configurazione del sistema. Il BDOS (Basic Disk Operating System), che gestisce i file su disco, e il BIOS (Basic Input/Output System), che contiene le routine di gestione delle periferiche, vengono conservati nella parte più alta della memoria. Sotto di essi si trova il Console Command Processor (CCP), ossia il modulo di comunicazione del CP/M.

All'estremità opposta della mappa di memoria ci sono 256 byte (detti "pagina zero") usati per registrare le variabili di sistema e altre informazioni di lavoro necessarie al funzionamento del CP/M. Alcune di esse hanno grande importanza, come gli "indirizzi d'ingresso" (entry point) per l'accesso alle aree di memoria del BDOS e del BIOS e al programma di caricamento ("bootstrap"). Tale programma deve essere conservato in memoria perché il sistema si trova spesso nelle condizioni di dover ricaricare il CP/M (il motivo di questa necessità viene spiegato più avanti).

Un altro importante modulo, situato alla pagina zero, è il cosiddetto Buffer di Controllo del File Transiente o TFBC. In precedenza si è visto che, se l'utente ordina al sistema di caricare un file dal disco, la CCP crea in memoria una copia del Blocco di Controllo del File. Quando il BDOS localizza un file, lo confronta con quello predisposto dal CCP, quindi comunica al CCP stesso ulteriori informazioni, tratte dalla traccia della directory, che vengono immagazzinate nel TFCB.

Tra la pagina zero e il modulo CCP rimane un'area di memoria chiamata Area del Programma Transiente, o TPA, ma più nota come area di lavoro del CP/M. Come si è già visto, quando il sistema operativo riceve un'istruzione, la CCP la confronta con l'elenco dei comandi residenti contenuti al suo interno. Se il confronto è negativo, la CCP interpreta l'istruzione come un comando "transiente" e ordina al BDOS di individuarlo. Supponendo che il comando risieda nel disco di sistema, il BDOS carica una copia del programma di codice macchina nell'Area del Programma Transiente, sempre a partire dall'indirizzo esadecimale 100 (l'inizio della "pagina uno" nella mappa di memoria del computer). Terminato il caricamento, il comando viene eseguito (ossia il programma viene fatto partire) automaticamente.

Per funzionare correttamente, il CP/M deve disporre di almeno 16 Kbyte di RAM, un'area di memoria piuttosto vasta considerando che il CP/M è stato progettato per operare in sottofondo mentre vengono eseguiti altri programmi. Fortunatamente solo una piccola parte di quest'area viene effettivamente usata dai programmi del CP/M. La maggior parte della RAM viene infatti assegnata alla TPA che, in versione minima, termina all'indirizzo hex 2900. Poiché il CP/M riserva la pagina zero alle variabili di sistema, l'Area del Programma Transiente contiene dunque 7 Kbyte.

Poiché la maggior parte dei comandi transienti non occupa più di 2 o 3 Kbyte, il resto della TPA viene usato per i file sui quali opera il comando transiente.

Tuttavia questa configurazione della memoria può causare inconvenienti. Se la TPA contiene 7 Kbyte liberi (RAM in versione minima) e il comando transiente ne occupa 3, restano solo 4 Kbyte a disposizione dei file: quanto basta per un programma di medie dimensioni, ma del tutto insufficienti per grossi file di testo (si ricordi che un file CP/M può raggiungere una lunghezza massima di 16 Kbyte). Naturalmente è possibile eliminare il problema introducendo nel sistema banchi di memoria RAM supplementare, che vengono assegnati alla TPA, fino al massimo di 64 Kbyte indirizzabili direttamente dal CP/M. L'indirizzo iniziale del CP/M viene quindi innalzato di 4000 (esadecimale) per ogni blocco supplementare di RAM da 16 Kbyte.

Si supponga tuttavia di non poter aggiungere la quantità di memoria necessaria. In tal caso, il file caricato supera la capacità dell'Area del Programma Transiente e va a sopra-scrivere l'area della CCP. Fortunatamente, durante l'esecuzione di un comando o di un programma, la CCP non è necessaria. Infatti, come molti altri sistemi operativi, il CP/M non accetta ulteriori comandi durante tale operazione e la presenza della CCP è pertanto ininfluente.
Ovviamente, prima di concludersi, il programma del comando transiente ricarica la sezione CCP per consentire al sistema di ricevere l'istruzione successiva. Tale operazione viene eseguita chiamando la routine di "ricaricamento del sistema" (in inglese "boot") situata alla locazione esadecimale 0005 di pagina zero. Questo è l'indirizzo d'entrata nel sistema operativo che ricarica in memoria il modulo CCP preparandolo a ricevere il comando successivo.

I limiti del CP/M

Cerchiamo adesso di capire la natura intrinseca dei comandi transienti. Quando Gary Kildall progettò il CP/M, l'hardware disponibile era ancora molto costoso (soprattutto i chip RAM) per cui solo pochi apparecchi possedevano più di 16 Kbyte. Di conseguenza il software sviluppato per queste macchine doveva adattarsi alle restrizioni imposte dall'hardware.
Poiché il CP/M puntava ad essere un sistema operativo il più possibile completo, pur mantenendosi all'interno della scarsa memoria disponibile, fu necessario assegnare un'area di memoria comune ad alcune delle sue parti. Questa tecnica, inapplicabile al BDOS e al BIOS che devono risiedere permanentemente in memoria, fu invece adottata per molti comandi che gestiscono l'accesso ai file e alcune operazioni d'ingresso e uscita (come l'invio di un file a una stampante o la sua visualizzazione sullo schermo). Fu deciso allora di assegnare la medesima area di memoria alla CCP e alla TPA perché questi moduli non vengono mai usati contemporaneamente dal CP/M.
L'area della CCP può quindi essere occupata da un programma transiente e, al termine della sua esecuzione, ricaricata con le routine del CCP.
Benché il mercato dei computer gestionali sia sempre più dominato da apparecchi con processori a 16 bit dotati di nuovi sistemi operativi come l'MS-DOS, il CP/M ed i micro Z80 a 8 bit non sembrano destinati ad un rapido declino.
Ai personal e agli home computer invece non occorre la potenza d'elaborazione dei processori a 16 bit, ma un buon sistema operativo per le loro unità a dischetti! Quindi il CP/M rappresenta ancora la scelta ideale per i fabbricanti di computer che intendono fornire una vasta base software senza accollarsi i costi di sviluppo. Alcune importanti compagnie hanno già imboccato questa strada: tra esse la Memotech e l'Amstrad.

Il "caso" Amstrad

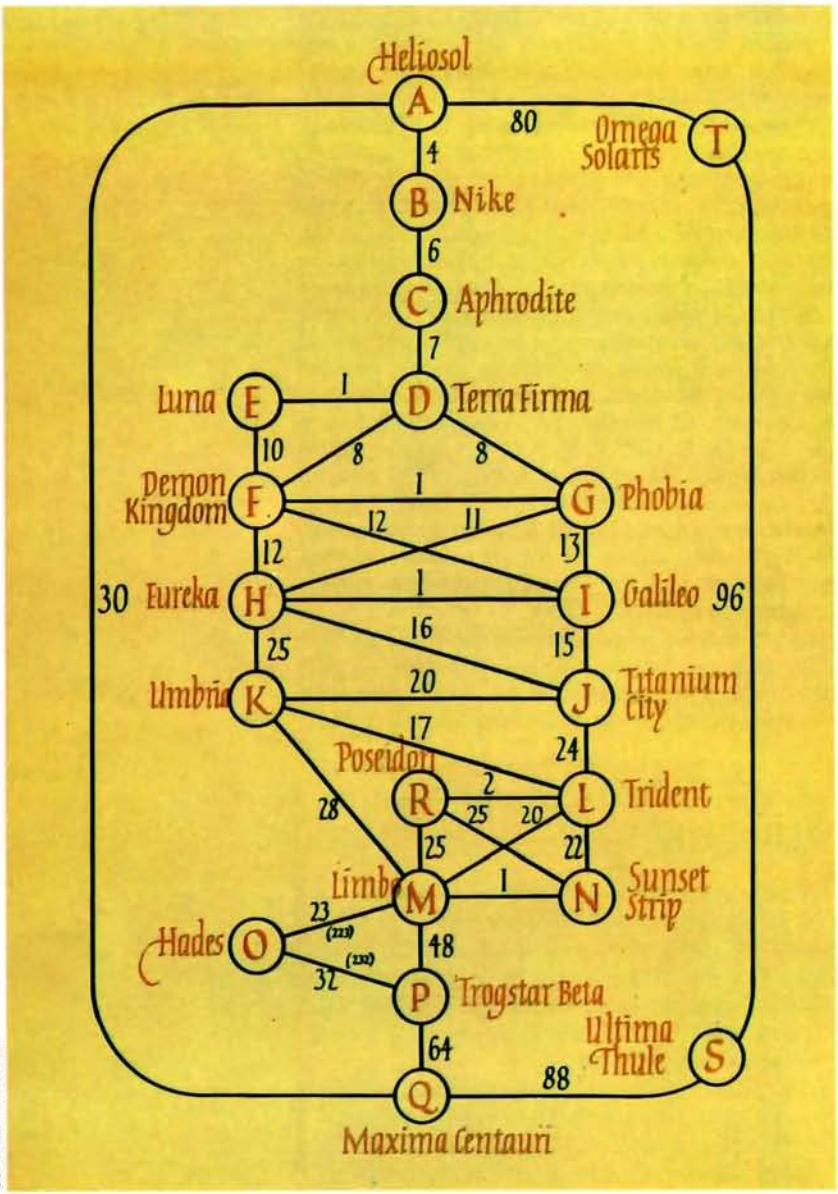
Adottando il CP/M, l'Amstrad ha messo a disposizione dei propri utenti più di 10.000 pacchetti software, ma ha creato loro numerosi problemi. Infatti il CP/M è stato sviluppato per dischetti da 8 e 5 1/4 pollici, mentre l'Amstrad ha optato per il formato Hitachi da 3 pollici. Quindi il numero dei pacchetti compatibili con il CP/M destinati agli utenti Amstrad è assai limitato.
Un secondo problema riguarda le funzioni di schermo dei computer Amstrad che lasciano a disposizione dei programmi solo 38 KByte di memoria, insufficienti per elaborare le più recenti applicazioni sviluppate sotto CP/M. Se tale software non verrà opportunamente modificato, i possessori di Amstrad saranno costretti a usare pacchetti un po' "antiquati".
La Digital Research non ha abbandonato il mercato dei 16 bit, tuttavia in questo settore ha incontrato una concorrenza molto più agguerrita di quella che contrastò il lancio del CP/M. Il CP/M-86 è il primo tentativo della Digital Research di inserirsi nel campo dei computer basati sui chip Intel 8086/8. Purtroppo questo CP/M a 16 bit ha incontrato pochi entusiasmi, in quanto il mercato ha preferito optare per la compatibilità IBM e per lo standard MS-DOS.
Recentemente la compagnia ha lanciato anche una versione multi-utente del CP/M, chiamata Concurrent CP/M, utilizzabile da un solo utente o da più computer in rete.

ALL'INTERNO DELL'MP/M E DEL CP/NET

In questa breve serie è stata trattata esclusivamente la versione 2.2 del CP/M, indubbiamente la più usata tra quelle disponibili. Questa, come tutti gli altri sistemi CP/M, è stata progettata per applicazioni mono-utente. Tuttavia la Digital Research ha sviluppato altri sistemi basati sul CP/M, detti "sistemi multitasking" (a più processi), utilizzabili contemporaneamente da più utenti.
L'unità centrale di elaborazione di un sistema multitasking ripartisce il proprio tempo d'elaborazione tra diversi utenti o periferiche. Per consentire loro di accedere alla stessa CPU è stato sviluppato un sistema operativo chiamato MP/M (Multi-Processing Monitor Control Program). Sotto questo sistema, un singolo computer può collegarsi ad un massimo di 16 terminali indipendenti tra loro. L'MP/M gestisce i rapporti con tutti gli utenti tramite una serie di funzioni che controllano l'accesso simultaneo ai file o alle periferiche del sistema come le stampanti.
Un altro sviluppo del CP/M prodotto dalla Digital Research è il CP/NET, un sistema operativo che collega in rete più utenti dotati di apparecchi differenti. Contrariamente all'MP/M, il CP/NET non richiede un'unità centrale allacciata a terminali subordinati anche se possiede un "nodo pilota" che controlla l'intera rete. Questo nodo pilota deve "girare" sotto MP/M ed essere collegato ad unità a dischetti

Codici genetici

Analisi dei processi evolutivi e soluzione di complessi problemi reticolari



BERNARD JENNINGS

Guardando le stelle

Questa immaginaria mappa interplanetaria mostra la rete di rotte iperspaziali tra i vari pianeti e i rispettivi tempi di viaggio. Il giocatore può visitare anche pianeti non direttamente collegati, ma ogni spostamento al di fuori della rete richiede 1000 unità di tempo. Il programma "GENE" ha il compito di trovare il percorso più breve su di una rete composta da 20 pianeti

Spesso gli studiosi dell'Intelligenza Artificiale si rifanno alla natura per progettare sistemi capaci di auto-migliorarsi; i sistemi biologici che più suscitano il loro interesse sono il sistema nervoso, il sistema immunitario e le teorie darwiniane sull'evoluzione. Il sistema nervoso, e in particolare il cervello umano, è uno "strumento" d'apprendimento di straordinaria efficienza, sebbene i meccanismi che ne regolano il funzionamento presentino ancora lati talmente oscuri da impedirne una vera e propria simulazione artificiale.

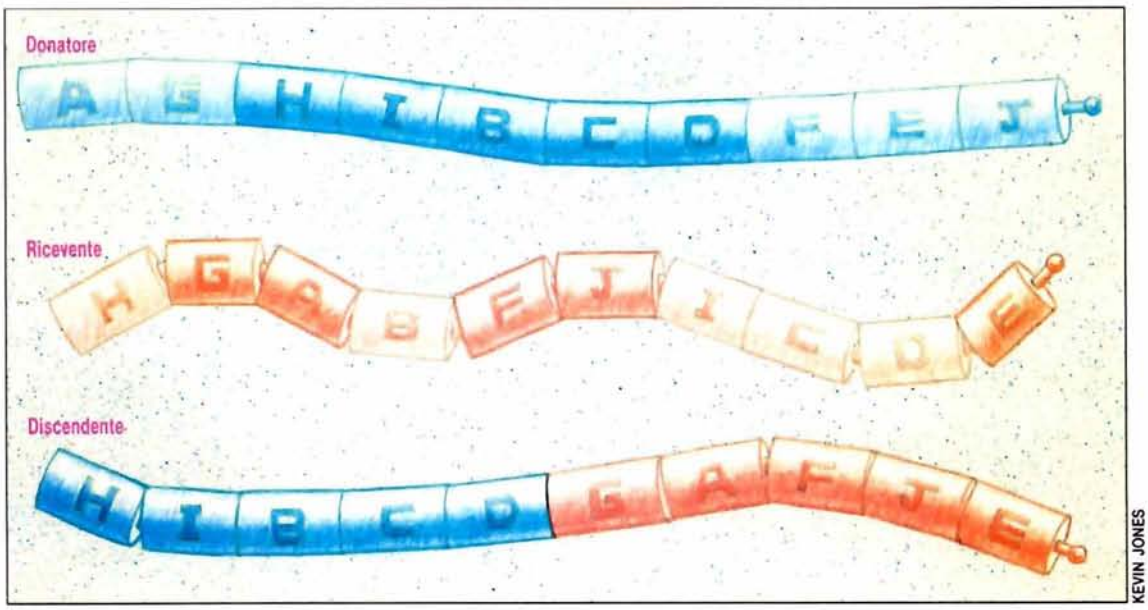
Si può dire che anche il sistema immunitario, ossia quel meccanismo di difesa capace di individuare e distruggere i corpi estranei che si introducono nell'organiso, sia uno strumento d'apprendimento; infatti, grazie alla sua straordinaria flessibilità, il sistema impara a riconoscere con precisione milioni di proteine diverse (antigeni) permettendo così all'organismo sopravvivere. Poiché i suoi meccanismi di apprendimento sono essenzialmente "meccanici", le capacità di generalizzazione del sistema immunitario sono un po' rudimentali.

Le teorie evoluzionistiche, infine, si basano su un meccanismo di selezione degli organismi molto progredito, ma estremamente lento. Questo processo può comunque essere facilmente accelerato da una simulazione computerizzata e applicato con buone probabilità di successo all'Intelligenza Artificiale. Si è detto che l'approccio darwiniano all'apprendimento da parte del computer offre, almeno in teoria, numerosi vantaggi. Vediamo adesso cosa accade nella pratica.

Per chiarire come funziona un algoritmo "evoluzionistico" applicato all'apprendimento da parte del computer, si può prendere ad esempio il famoso rompicapo chiamato "problema del commesso viaggiatore": durante un viaggio d'affari, un commesso viaggiatore deve visitare, per una sola volta, le 48 capitali degli Stati Uniti continentali (escludendo Alaska e Haway) e infine tornare al punto di partenza. Usando una tabella delle distanze tra le città, occorre programmare il viaggio in modo da ridurne al minimo la lunghezza. A prima vista il problema sembra semplice ma, in realtà, il numero di percorsi alternativi è (N-1)! dove N rappresenta il numero delle città. In un giro che prevede 48 città, le prime fermate possibili sono ben 47, le seconde 46, le terze 45 e così via. Quindi, anche dovendo toccare solo 20 citta, i percorsi alternativi sarebbero molti miliardi!

Questo rompicapo può essere affrontato in molti modi: lo si può trattare come un problema di ricerca (ricorrendo a tecniche simili a quelle descritte a pag. 1721) oppure applicando metodi di ricerca casuali, come quello chiamato "Monte Carlo". Una terza alternativa, certamente meno ortodossa, è quella di considerarlo un problema di apprendimento e di usare un algoritmo che esplori in modo selettivo l'immenso numero di soluzioni alternative. La soluzione ottenuta non sarà forse quella ottimale, ma certo una delle migliori possibili!

Il programma *GENE* (General Evolutionary



KEVIN JONES

Nuove generazioni

Il programma "GENE" crea nuovi percorsi all'interno della rete partendo da due percorsi preesistenti. Questi sono rappresentati da stringhe BASIC di 20 caratteri, che vengono ricombinate per dar vita ad una stringa "discendente". Nell'esempio a fianco, la sottostringa HIBCD, tratta dalla stringa "donatore", viene combinata a quella "ricevente" per generare una nuova stringa nella quale non compaiono lettere identiche. Tale procedimento, simile a quello della riproduzione sessuata, permette di trasferire ai "discendenti" le caratteristiche dei genitori

Network Explorer), qui presentato, è un sistema di apprendimento adattato appositamente per esplorare reti logiche e per definire regole in continua evoluzione, e che permette di trovare percorsi sempre più brevi all'interno della rete. Prima di descriverne il funzionamento, è necessario creare la rete da far esplorare al programma e un buon esempio è fornito dal gioco *The Gatecrasher's Guide To The Galaxy* in cui vengono elencati, tra l'altro, tutti i pianeti, nel raggio di 80 anni luce dalla Terra, sui quali, la domenica sera, vengono tenute divertenti feste. Grazie a questa mappa, il precedente rompicapo può essere trasformato in un gioco astronomico nel quale il giocatore deve visitare in una sola notte tutti e 20 i pianeti e tornare alla base di partenza.

La mappa mostra le rotte iperspaziali che collegano i diversi astri e il tempo necessario per spostarsi da un nodo all'altro. I viaggi tra nodi non collegati da linee iperspaziali hanno una durata fissa di 1.000 unità di tempo.

Schema di apprendimento evoluzionistico

Un classico schema di apprendimento evoluzionistico contiene una serie di strutture o "regole" che vengono trattate come "pseudo-organismi". Ciascuna regola definisce una potenziale soluzione del problema affrontato. Inoltre le regole vengono usate anche per generare nuove strutture (dette "discendenti"), con una tecnica in parte simile alla riproduzione biologica, nella quale i genitori forniscono ai figli parte del loro patrimonio genetico.

Basandosi sui risultati ottenuti, il programma seleziona quelle regole che hanno maggiori probabilità di "sopravvivere" e di generare discendenti.

La struttura delle regole/organismi di GENE, contenute in forma di stringhe nel program-

INGEGNERIA... ALFABETICA

Il programma "GENE" sceglie a caso due strutture "genitrici" sopravvissute al processo di selezione (pertanto "geneticamente" superiori al livello medio della popolazione di stringhe): una viene chiamata "donatore" (R1%), l'altra "ricevente" (R2%) (vedere le linee 3030-3060).

Quindi dal donatore viene estratto, in modo casuale, un segmento di "materiale genetico" che viene poi assegnato alla stringa S$. Infine viene chiamata la subroutine alla linea 3300, che unisce il segmento alla stringa ricevente ed elimina da essa i caratteri duplicati, lasciando inalterato l'ordine degli altri. Questo processo di ricombinazione non è simmetrico: infatti si ottengono risultati diversi combinando X a Y oppure Y a X anche se le posizioni casuali P1% e P2% sono identiche.

Ecco un esempio in scala ridotta. Date le due stringhe "genitrici":

Donatore: A G H I B C D F E J
Ricevente: H G A B F J I C D E

è possibile estrarre la sottostringa

H I B C D

dal donatore e combinarla alle lettere del ricevente non eliminate per generare la stringa discendente

H I B C D G A F J E

Naturalmente esistono altre tecniche che permettono di ricombinare una coppia di regole/stringhe, ma tutte devono garantire che, come accade in un vero incrocio genetico, i segmenti d'informazione forniti dai genitori raggiungano inalterati la generazione successiva e che ogni ricombinazione produca un discendente "valido". Se queste condizioni vengono rispettate, la routine di ordinamento, inserita alla linea 4000 per completare il programma GENE, non svolge alcuna funzione

ma BASIC, è abbastanza semplice. Ad esempio, nella stringa:

ABJHMNCDKTSFRQEGILOP

ogni lettera indica un corpo celeste della rete planetaria e compare una sola volta nella stringa. Pertanto ogni stringa di 20 caratteri rappresenta una permutazione di pianeti da visitare e definisce un possibile percorso attraverso la rete.
Ogni stringa viene valutata sommando le distanze necessarie per toccare tutti i pianeti nell'ordine indicato: il percorso migliore è, ovviamente, quello più breve.
Le stringhe che producono risultati superiori alla media vengono eliminate dopo ogni generazione e sostituite da nuove stringhe. La maggior parte di queste vengono create con un procedimento simile a quello della riproduzione sessuata, mentre circa l'8% di esse derivano da mutazioni delle stringhe sopravvissute.
La subroutine Mutazione, che inizia alla linea 3500, esegue un certo numero di sostituzioni casuali. Pur non essendo i principali "operatori genetici", le mutazioni introducono un meccanismo secondario di ricombinazione che consente al sistema di non fermarsi al miglior risultato ottenuto con la semplice riproduzione sessuata.
Durante l'esecuzione del programma, si può notare che la qualità media della popolazione delle regole migliora sensibilmente col passare del tempo, sebbene ciò non accada con una progressione uniforme. Il programma, infatti, può commettere errori nel tentativo di sostituire regole buone con altre migliori.
Tuttavia, è possibile calibrare il sistema giocando sul valore del tasso di decesso alla linea 2520. L'elemento casuale di questa linea seleziona l'ottantotto per cento come indice di sopravvivenza per le regole che passano alla generazione successiva, ma questo parametro può essere modificato. Inoltre è possibile cambiare la frequenza di mutazione alla linea 3520 o introdurre fattori di mutazione alternativi come la completa inversione di una regola/stringa.
Per concludere, si può tentare una valutazione obiettiva del metodo adottato e del suo criterio di funzionamento. Il metodo adottato può essere analizzato confrontandolo con il sistema Monte Carlo: tutti gli algoritmi genetici sono, in effetti, procedure Monte Carlo modificate. Un metodo Monte Carlo "puro" genera soluzioni casuali e seleziona la migliore in un intervallo di tempo dato. Nel nostro esempio, invece, non ci sono limiti di tempo per creare una regola/stringa casuale, valutarla, conservarla se è la migliore generata sino a quel momento e sottoporla a mutazione.
Inoltre il programma riesce a trovare in breve tempo una buona soluzione, che subisce solo lievi miglioramenti via via che il tempo passa: una soluzione ottenuta dopo 20.000 tentativi, dunque, non si discosta molto da quella trovata dopo 10.000. In linea di principio, l'applicazione prolungata di un algoritmo genetico migliora la qualità del responso e ciò spiega, almeno in parte, quale sia il funzionamento del metodo.
Un metodo Monte Carlo "puro" adotta un sistema di ricerca "cieco", mentre un algoritmo genetico usa le soluzioni trovate per pilotare la ricerca successiva. Le configurazioni che ne migliorano le prestazioni vengono conservate, propagate attraverso la base di conoscenza (la popolazione di regole) e ricombinate per generarne altre leggermente diverse. In effetti la ricerca viene indirizzata principalmente verso problematiche di tipo spaziale (multi-dimensionali) per le quali è possibile sviluppare efficienti strategie di ricerca, purché la funzione di valutazione non sia eccessivamente discontinua.

IL PROGRAMMA GENE

Le principali strutture di dati per questo sistema d'apprendimento evoluzionistico sono le seguenti matrici e variabili:

NOME$(MIS%)	Nomi dei nodi della rete
DIST%(MIS%,MIS%)	Distanze tra i nodi
NX%(MIS%)	Usata nella routine di ricombinazione
R$(NR%)	Le regole (stringhe)
RV(NR%)	I valori delle regole impiegate

NX% - Usata per aggiungere il segmento fornito dal ricevente quando viene generata una nuova regola (routine 3300) in modo che ogni lettera/nodo compaia una sola volta nel "discendente".
R$ - Contiene la popolazione corrente di regole/stringhe di lunghezza LUNG%. Ogni stringa definisce un particolare percorso.
RV - Fornisce il "valore" di ogni regola, corrispondente alla lunghezza del percorso. Se RV(R%)=0, la regola R% viene eliminata e deve essere sostituita nella generazione successiva. (Ovviamente non sono ammessi percorsi con valore 0.)
MIS% - Può essere modificata per operare su reti di diverso tipo. In tal caso vanno cambiate le linee dalla 8000 in poi ed, eventualmente, la variabile NR% per osservare gli effetti di una maggiore o minore popolazione di regole.

```
  10 REM ************
  11 REM **  GENE  **
  13 REM ************
 100 GOSUB 1000: REM crea la matrice della mappa
 101 G%=0: SN%=0
 105 B=MIS%*1000: B$=""
 110 INPUT "Quante generazioni? ", MG%
 111 GOSUB 1700: REM regole iniziali
 115 IF SN%=0 THEN INPUT "Il nodo di partenza e' il
     N. ", SN%
 120 REM ** CICLO PRINCIPALE DEL PROGRAMMA **
 130 G%=G%+1
 140 PRINT "Generazione "; G%
 150 GOSUB 2000: REM valuta le regole
 160 GOSUB 2500: REM elimina le regole peggiori
 170 GOSUB 3000: REM combina quelle migliori
 180 GOSUB 3500: REM mutazioni
 190 GOSUB 4000: REM ordinamento
 200 IF G%<MG% THEN 120
 220 GOSUB 5000: REM visualizza le nuove regole
 250 END
 999 :
1000 REM -- CREA LA MAPPA A RETE:
1001 MIS%=20: NR%=24
1010 DIM NOME$(20), DIST%(20,20)
1011 DIM NX%(MIS%)
1012 DIM R$(NR%), RV(NR%)
1013 REM regole e loro valori
1015 FOR I%=1 TO MIS%
1020 FOR J%=1 TO MIS%
1022 DIST%(I%,J%)=1000: REM valore per difetto
1023 IF I%=J% THEN DIST%(I%,J%)=0
1025 NEXT: NEXT
1030 NC%=0: L%=0
1032 FOR I%=1 TO NR%: RV(I%)=
1033 RESTORE
1040 REM ** LEGGE NOME E NUM.
1050 READ N$,ID%
1055 PRINT N$,ID%
1060 IF ID%<>0 THEN GOSUB 15
1070 IF ID%<>0 THEN 1040
1080 PRINT NC%;" LOCAZIONI M
1088 PRINT L%;" collegamenti
1090 RETURN
1100 :
1500 REM -- NODI INDIVIDUALI
1510 NC%=NC%+1
1520 IF ID%<>NC% THEN PRINT
     " fuori ordine."
1530 NEME$(NC%)=N$
1550 REM ** CREA MATRICE DIST
1560 READ NI%, NT%
1570 DIST%(ID%,NI%)=NT%
1588 REM trascura i collegame
1590 L%=L%+1
1600 IF NI%>0 THEN 1550
1610 RETURN
1620 :
1700 REM -- Regole iniziali
1710 S$=LEFT$ ("ABCDEFGHIJKL
1715 GOSUB 1780: REM legge il
1720 FOR R%=1 TO NR%
1730 R$(R%)=S$
1740 PRINT S$; R%
1750 I%=INT(RND(1)*MIS%+1):
1760 GOSUB 6000: REM SCAMBIA
1770 NEXT
1775 RETURN
1777 :
1780 INPUT "Vecchio File Regc
     RF$
1800 F%=OPENUP (RF$)
1810 INPUT #F%, B$, B, SN%
1820 CLOSE #F%
1825 S$=B$
1830 REM solo quello piu' in
1840 RETURN
1850 :
2000 REM -- Routine di valut
2010 T=0
2020 FOR R%=1 TO NR%
2030 IF RV(R%)<=0 THEN GOSUB
2040 T=T+RV(R%)
2050 NEXT
2060 AV=T/NR%: REM valore med
2070 PRINT "Valore medio = "
2080 RETURN
2100 :
2200 REM -- Valutazione di un
2210 S$=R$(R%)
2220 P1%=SN%: REM nodo di par
2230 GT%=0
2240 FOR S%=1 TO LEN(S$)
2250 P2%=ASC(MID$(S$,S%,1))-
2260 IF P2%=SN% THEN GOTO 225
2270 GT%=GT%+DIST%(P1%,P2%)
2280 P1%=P2%
2290 NEXT
2300 RV(R%)=GT%+DIST%(P2%,SN%
2310 RETURN
2320 :
2500 REM -- Routine che elimi
     peggiori:
2510 FOR R%=1 TO NR%
2515 IF RV(R%) < B THEN B=RV(
2520 IF RV(R%) > AV OR INT(RN
     R$(R%)="": RV(R%)=0
2530 NEXT
2540 RETURN
2550 REM -- I valori piu' bas
2560 :
3000 REM -- Routine di ricomb
3010 FOR R%=1 TO NR%
3020 IF RV(R%)>0 THEN GOTO 31
3030 R1%=INT(RND(1)*NR%+1): I
3050 R2%=INT(RND(1)*NR%+1): I
3070 REM 'genitori' scelti a
3075 P2%=INT(RND(1)*(MIS%-1)+
3080 P1%=INT(RND(1)*MIS%+1):
3090 S$=MID$(R$(R1%),P1%,P2%)
3100 GOSUB 3300: REM collega
3110 R$(R%)=S$
3120 NEXT
3140 RETURN
3150 :
3300 REM -- Routine di connes
3310 FOR S%=1 TO MIS%
3320 NX%(S%)=0: NEXT
3330 FOR S%=1 TO LEN(S$)
3340 SX%=ASC(MID$(S$,S%,1))-
3350 NX%(SX%)=NX%(SX%)+1
3360 NEXT
3370 FOR S%=1 TO LEN(R$(R2%))
3380 SX%=ASC(MID$(R$(R2%),S%,
3390 IF NX%(SX%) >0 THEN GOTO
3400 S$=S$+MID$(R$(R2%),S%,1)
3410 NX%(SX%)=NX%(SX%)+1
```

```
: NEXT
)EL NODO **
)RIZZATE."
)lla rete."
 COLLEGAMENTI:
'TENZIONE: Nodo ";ID%;
 TRA NODI **
:i-zero
ttizie:
)PQRSTUVWXYZ", MIS%)
file (se presente)
=INT(RND(1)*MIS%+1)
)E CARATTERI
 (RETURN se assente)",
lto
ione della regola
200
)
)V
 sola regola:
enza
 le regole
%): B$=R$(R%)
(1)*100) >88 THEN
 sono i migliori.
nazione:
)
 RV(R1%) <=0 THEN 3030
 RV(R2%) <=0 THEN 3050
aso.
)
EM punti di giunzione
 caratteri rimasti
ine del Gene!
))-64
3420
3420 NEXT
3440 RETURN
3450 :
3500 REM -- Routine di Mutazione:
3510 FOR R%=1 TO NR%
3520 IF RND(100) > 8 THEN GOTO 3580
3522 S$=R$(R%)
3525 FOR T%=1 TO 7
3530 R1%=INT(RND(1)*MIS%+1)
3540 R2%=INT(RND(1)*MIS%+1)
3560 I%=R1%: J%=R2%: GOSUB 6000: REM SCAMBIA DUE
     CARATTERI
3565 NEXT T%
3570 R$(R%)=S$
3575 RV(R%)=0
3580 NEXT
3590 REM unico scambio effettuato.
3595 REM ora e' necessaria un'inversione.
3600 RETURN
3620 :
4000 REM -- Routine di ordinamento
4010 RETURN
4020 REM -- linea di comodo (per il momento)
4040 :
5000 REM -- Visualizza i risultati:
5010 PRINT'"I percorsi sono :"
5020 BR=MIS%*1000
5030 R%=0
5040 FOR I%=1 TO NR%
5050 IF RV(I%)=0 THEN 5100
5060 PRINT I%,RV(I%)
5070 PRINT R$(I%),
5080 IF RV(I%)< BR THEN BR=RV(I%): R%=I%
5100 NEXT
5105 A$=GET$
5110 PRINT
5120 PRINT "Il migliore e' :"
5130 PRINT R$(R%),R%
5131 S$=R$(R%): GOSUB 6400
5133 PRINT "Distanza ="; RV(R%)
5134 A$=GET$
5135 PRINT'"La migliore e' :"
5136 PRINT B$
5140 S$=B$: GOSUB 6400
5143 PRINT "Distanza = ";B
5144 A$=GET$
5145 PRINT
5148 GOSUB 5500: REM visualizza il file
5150 RETURN
5160 :
5500 REM -- Routine di visualizzazione:
5510 INPUT "Nuovo File Regole (RETURN se assente)", RF$
5520 IF RF$="" THEN RETURN
5530 F%=OPENOUT(RF$)
5540 PRINT #F%, B$, B, SN%
5550 CLOSE #F%
5555 REM solo la migliore
5560 RETURN
5570 :
6000 REM ** SCAMBIA DUE CARATTERI IN S$ **
6040 IF I%>J% THEN T%=I%: I%=J%: J%=T%
6050 X$=MID$(S$,I%,1)
6060 Y$=MID$(S$,J%,1)
6070 S$=LEFT$(S$,I%-1) + Y$ + MID$(S$,I%+1)
6080 S$=LEFT$(S$,J%-1) + X$ + MID$(S$,J%+1)
6090 RETURN
6100 :
6400 REM ** VIAGGIO **
6430 PRINT "  0  "; NOME$(SN%)
6440 FOR I%=1 TO LEN(S$)
6450 N%=ASC(MID$(S$,I%,1))-64
6460 IF N%<>SN% THEN PRINT I%;" "; NOME$(N%)
6470 NEXT
6480 PRINT I%;" "; NOME$(SN%)
6490 RETURN
6500 :
8000 REM -- ISTRUZIONI DATA PER MAPPA PLANETARIA:
8010 DATA HELIOSOL, 1
8020 DATA 2,4, 17,30, 20,80, 0,0
8030 DATA NIKE, 2
8040 DATA 1,4, 3,6, 0,0
8050 DATA APHRODITE,3
8060 DATA 2,6, 5,7, 0,0
8070 DATA LUNA, 4
8080 DATA 5,1, 6,10, 0,0
8090 DATA TERRA FIRMA, 5
8100 DATA 3,7 7,8, 6,8, 4,1, 0,0
8110 DATA DEMON KINGDOM, 6
8120 DATA 5,8, 7,1, 9,12, 8,12, 5,10, 0,0
8130 DATA PHOBIA, 7
8140 DATA 5,8, 9,13, 8,11, 6,1, 0,0
8150 DATA EUREKA, 8
8160 DATA 6,12, 7,11,'9,1, 10,16, 11,25, 0,0
8170 DATA GALILEO, 9
8180 DATA 10,15, 8,1, 6,12, 7,13, 0,0
8190 DATA TITANIUM CITY, 10
8200 DATA 9,15, 12,24, 11,20, 8,16, 0,0
8210 DATA UMBRIA, 11
8220 DATA 8,25, 10,20, 12,17, 13,28, 0,0
8230 DATA TRIDENT, 12
8240 DATA 10,24, 14,22, 13,20, 11,17, 18,2, 0,0
8250 DATA LIMBO, 13
8260 DATA 12,20, 14,1, 16,48, 15,23, 11,28, 0,0
8270 DATA SUNSET STRIP, 14
8280 DATA 12,22, 13,1, 18,25, 0,0
8290 DATA HADES, 15
8300 DATA 13,223, 16,232, 0,0
8310 DATA TROGSTAR BETA, 16
8320 DATA 13,48, 17,64, 15,32, 0,0
8330 DATA MAXIMA CENTAURI, 17
8340 DATA 16,64, 1,30, 19,88, 0,0
8350 DATA POSEIDON, 18
8360 DATA 12,2, 14,25, 13,25, 0,0
8370 DATA ULTIMA THULE, 19
8380 DATA 17,88, 20,96, 0,0
8390 DATA OMEGA SOLARIS, 20
8400 DATA 1,80, 19,95, 0,0
8410 DATA NOWHERE, 0
```

Basic & Basic

Il programma "GENE" è stato scritto per il Micro BBC.

Commodore 64:

Inserire le seguenti modifiche:

```
1820 CLOSE 2
5105 GET A$:IF A$ = "" THEN 5105
5134 GET A$:IF A$ = "" THEN 5134
5144 GET A$:IF A4 = "" THEN 5144
5550 CLOSE 2
```

Aggiungere queste linee per i sistemi a disco:

```
1800 OPEN 2,8,2,RF$ + "S,R":F% = 2
5530 OPEN 2,8,2,RF$ + "S,W":F% = 2
```

oppure le seguenti per i sistemi a cassetta:

```
1800 OPEN 2,1,0,RF$:F% = 2
5520 OPEN 2,1,1,RF$:F% = 2
```

Spectrum:

Eliminare il simbolo % da tutte le variabili; sostituire NOME$( ) con N$, DIST%(,) con L(,), NX%( ) con N( ), RV( ) con R( ), MIS% con SI e RF$ con F$; racchiudere tra virgolette i nomi dei pianeti alle linee DATA. Apportare le seguenti modifiche:

```
1010 DIM N$(20,15),L(20,20)
1012 DIM R$(NR,20),R(NR),D$(3,25)

1710 LET S$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ",(TO SI)

1800 LOAD F$ DATA D$( )
1810 LET B$ = D$(1)
1820 LET B = VAL(D$(2)):LET SN = VAL(D$(3))

2250 LET P2 = CODE S$(S TO S)—64
3090 LET S$ = R$(R1)(P1 TO P2—P1)
3340 LET SX = CODE S$(S TO S)-64
3400 LET S$ = S$ + R$(RS)(S TO S)

5105 IF INKEY$ = "" THEN GO TO 5105
5134 IF INKEY$ = "" THEN GO TO 5134
5144 IF INKEY$ = "" THEN GO TO 5144

5530 LET D$(1) = B$: LET D$(2) = STR$(B)
5540 LET D$(3) = STR$(SN)
5550 SAVE F$ DATA D$( )

6050 LET X$ = S$(I TO I)
6060 LET Y$ = S$(J TO J)
6070 LET S$ = S$(T0 I—1) + Y$ + S$(I + 1 TO)
6080 LET S$ = S$(TO J—1) + X$ + S$(J + 1 TO)

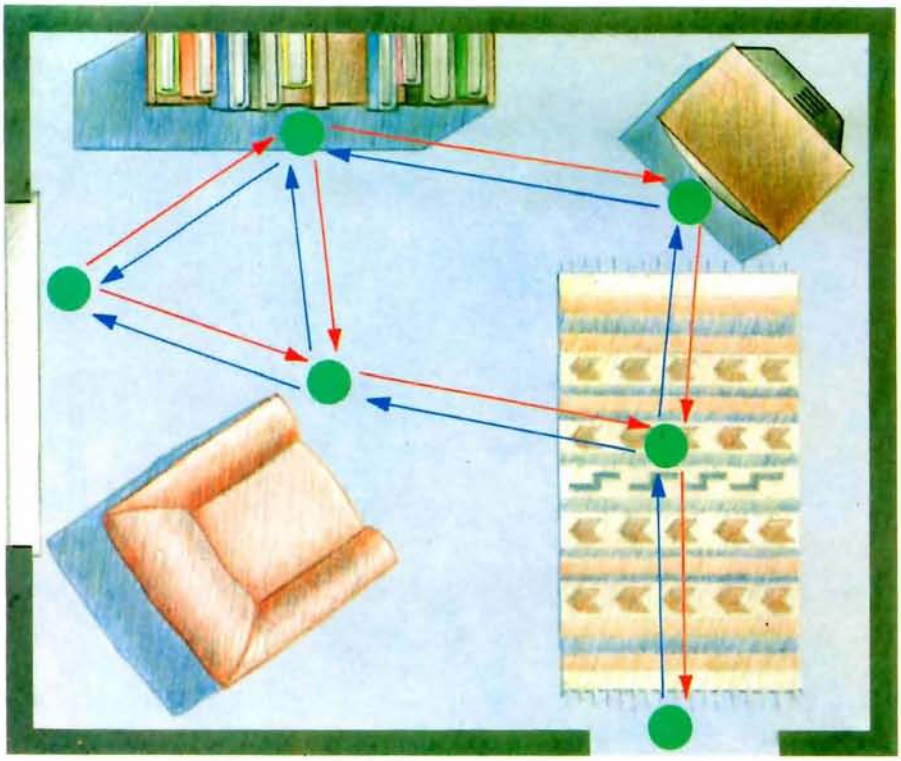
6450 N = CODE S$(I TO I)—64
```

Analisi introspettiva

Autonomia di movimento

La stanza dove si muove il robot contiene una serie di posizioni "note" (registrate come "fatti"). Queste sono collegate da una serie di sentieri da percorrere, memorizzati secondo un particolare formato ("percorso(tv, libreria)", "percorso(tappeto, sedia)", ecc.). Il programma a pagina seguente mostra come il robot adoperi queste informazioni per spostarsi da A a B

Il PROLOG può usare i programmi come dati: il programma per robot



I motivi che hanno fatto scegliere il PROLOG come linguaggio di base per i computer della "quinta generazione" sono principalmente due.
Per prima cosa gli elementi del PROLOG possono presentare un formato abbastanza simile alle "relazioni" di un sofisticato database relazionale, che gli esperti nipponici propongono come nucleo centrale del sistema da loro ideato.
Secondariamente il PROLOG è un linguaggio ideale per scrivere programmi relativi all'Intelligenza Artificiale, in quanto gli stessi programmi possono essere interpretati come dati e l'interprete/esecutore somiglia ai "motori d'inferenza" dei più recenti sistemi esperti (vedere alla pagina 1781).
Per comprendere i vantaggi offerti dall'uso delle istruzioni di programma come dati, basta esaminare un semplice programma di simulazione nel quale vengono adoperate alcune delle più sofisticate funzioni del PROLOG per controllare i movimenti di un robot in un ambiente delimitato (una stanza).
Al suo interno, il programma individua una serie di posizioni collegate da "sentieri" rettilinei. Il robot deve eseguire comandi del tipo: «spostati dal televisore alla sedia» scegliendo autonomamente il percorso più breve. Un comando del tipo vai(Luogo), significa: «spostati dalla posizione attuale a Luogo seguendo il tragitto più breve».
Per questa istruzione il programma prevede due clausole. La prima riguarda l'eventualità che il robot si trovi già nella posizione Luogo. Il database deve quindi possedere informazioni o "fatti" del tipo (a(porta)), che registra la posizione corrente del robot.
La seconda clausola usa tale fatto per stabilire la posizione di partenza e chiama un'altra procedura: vai(Posto1,Posto2).
Poiché questa procedura può essere usata da vai(Luogo) in modo autonomo, deve innanzi tutto controllare che Partenza e Luogo siano posizioni note e che Partenza corrisponda effettivamente alla posizione corrente. Se una delle due verifiche fallisce, il PROLOG abbandona la clausola considerata e passa a quella successiva. La clausola vai(Partenza,Luogo) sposta il robot in Partenza, se già non occupa quella posizione, chiamando in successione vai(Partenza) e vai(Luogo); questa procedura significa: «posizionati su Partenza, se già non occupi tale posizione, quindi raggiungi Luogo».
La terza clausola per vai(Partenza,Luogo) riguarda la visualizzazione di un messaggio d'errore se nessuna delle due posizioni è nota.

Il predicato "findall"

Se le verifiche preliminari per la prima clausola sono risultate positive, il successivo obiettivo del programma è findall, un predicato che può essere introdotto dall'utente qualora non sia "incorporato" nel PROLOG (come il FORTH, anche il PROLOG può essere esteso definendo nuovi predicati che assumono lo stesso rango di quelli di sistema).
Il predicato findall prevede tre argomenti: il primo è un nome di variabile, il secondo uno scopo e il terzo una lista (RL).
Il suo obiettivo è cercare di dimostrare lo scopo assegnatogli, in questo caso traccia(Partenza,Luogo,[ ],Tragitto).
La variabile relativa al primo argomento deve corrispondere a una delle variabili dello scopo e, quando ciò si verifica, il valore assegnatole viene aggiunto alla lista nel terzo argomento.
Per esempio, lo scopo traccia(Posto1, Posto2,[ ],Tragitto) dovrebbe definire un percorso tra Posto1 e Posto2 e registrarlo nella variabile Tragitto.
Si noti che non c'è alcun bisogno di definire una procedura per findall; infatti, grazie allo stile "dichiarativo" dei programmi PROLOG, le procedure vengono scritte secondo un formato "discendente" (top-down), definendo prima gli scopi di livello più alto e specificando i dettagli successivamente.
Un altro predicato, non ancora definito, è accorcia(RL,TragittoBreve) che ha il compito di ispezionare una lista di liste

(Lista1) e crearne una nuova (Lista2) contenente solo la lista più breve.
Poiché tutte le liste contenute in Lista1 sono dei percorsi, la lista più breve rappresenta ovviamente il tragitto più breve.
Individuato il tragitto più breve per raggiungere la posizione d'arrivo, il robot viene mosso lungo questo percorso, prescelto per mezzo di una "falsa" procedura percorri(Tragitto) che traccia il percorso, ossia una lista di scopi PROLOG che, invece di essere visualizzata sullo schermo, viene eseguita dal PROLOG come un programma. Infine vengono definiti il predicato ruota(Posto), che individua la posizione d'arrivo e ruota il robot verso quella direzione, e la procedura muovi(Posto1,Posto2) che permette al robot di spostarsi da una posizione all'altra.
Il programma per il movimento viene sviluppato dal programma principale all'interno della procedura traccia, usando il seguente algoritmo: «per tracciare un percorso da A a B, trova prima un sentiero d A a C, poi da C a B».
La procedura traccia è ricursiva. La prima clausola prevede l'interruzione della ricursione quando viene raggiunta la posizione d'arrivo.
La seconda clausola, che in pratica svolge tutto il lavoro, dichiara che è possibile muovere il robot da A a B lungo il percorso R se:

1. esiste un sentiero da A a C e (AND)
2. C non fa parte della lista V già esaminata (C viene aggiunto alla lista per evitare che in seguito il robot si muova lungo percorsi circolari), e (AND)
3. esiste un tracciato per spostarsi da C a B lungo il sentiero R1.

Se tutte queste condizioni sono vere, il sentiero finale R è la lista contenente un programma per spostarsi da A a C sommato al cammino sin qui percorso (contenuto in R1).
La logica di queste procedure ricursive è piuttosto complessa, ma consente al PROLOG una straordinaria concisione.
Per completare il programma è necessario aggiornare la posizione del robot usando i due predicati di sistema retract e assert.
Retract(X) elimina dal database la prima clausola che "collima" con X, e asserta(X) introduce la clausola X nel database come prima clausola di quel tipo (mentre assertz(X) aggiunge X come ultima clausola).
Questi predicati sono strumenti essenziali per la programmazione in Intelligenza Artificiale.

Simulazione del robot "semovente"
Versione PROLOG standard:

```
pos(porta).              /*la posizione corrente del robot*/

vai(Luogo):-             pos(Luogo), write('Occupo già questa posizione!'),nl,nl,nl.
vai(Luogo):-             pos(Partenza), vai(Partenza,Luogo).

vai(Partenza,Luogo):-    posto(Partenza), posto(Luogo), pos(Partenza), findall(Tragitto, traccia(Partenza,Luogo,< >,Tragitto),RL), accorcia(RL,TragittoBreve), percorri(TragittoBreve), retract(pos(Partenza)), asserta(pos(Luogo)), posizione.
vai(Partenza,Luogo):-    non(pos(Partenza)),write('Non mi trovo a'), write(Partenza), write('quindi devo prima raggiungere quella posizione.'),nl,nl, vai(Partenza), vai(Luogo).
vai(Partenza,Luogo):-    write('Posso raggiungere solo posizioni note'),nl,nl.

traccia(A,A,_,R).
traccia(A,B,V,R):-       percorso(A,C), non(member(C,V), append(<C>,V,V1), traccia (C,B,V1,R1), append(<ruota(C), muovi(A,C)>,R1,R).

percorri(Tragitto):-     write(Tragitto),nl, nl, nl. /*da definirsi in seguito*/

posizione:-              pos(Posto), write('Mi trovo al'), write(Posto),nl,nl,nl.

/*lista dei sentieri conosciuti dal robot*/   /*lista delle posizioni note*/

percorso(libreria,sedia).          posizione(porta).
percorso(sedia,libreria).          posizione(tappeto).
percorso(libreria,tv).             posizione(tv).
percorso(tv,libreria).             posizione(libreria).
percorso(tv,tappeto).              posizione(finestra).
percorso(tappeto,tv).              posizione(sedia).
percorso(tappeto,sedia).
percorso(sedia,tappeto).
percorso(porta,tappeto).
percorso(tappeto,porta).
percorso(finestra,sedia).
percorso(sedia,finestra).
percorso(finestra,libreria).
percorso(libreria,finestra).
```

Versione Micro-PROLOG:

```
pos(porta)               /*la posizione corrente del robot*/

((vai X)                 (pos X)(P Occupo già questa posizione!)PP,PP,PP)
((vai X)                 (pos Y)(vai XY))

((vai XY)                (posto X)(posto Y)(pos X)
                         (findall Z(traccia XY( )Z)x)
                         (accorcia xy)
                         (percorri y)
                         (DELCL((pos X)))(ADDCL((pos Y)))
                         (posizione))
((vai XY)                (NOT pos X)
                         (P Non mi trovo a)(PX)
                         (P quindi devo prima raggiungere quella posizione.)PP PP
                         (vai X)(vai Y))
((vai XY)                (P Posso raggiungere solo posizioni note)PP PP)
(traccia XXx1Z)
((traccia XYxZ)          (percorso X X1)(Not member X1 x)(append(X1) x x1)
                         (traccia X1 Y x1 Z1)
                         (append((ruota X1)(muovi X X1))Z1 Z))

((percorri Z)            (P Z)PP PP)
((posizione)             (pos X)(P Mi trovo a X)PP PP PP)

/*lista dei sentieri conosciuti dal robot*/   /*lista delle posizioni note*/

(percorso libreria sedia)          (posizione porta)
(percorso sedia libreria)          (posizione tappeto)
(percorso libreria tv)             (posizione tv)
(percorso tv libreria)             (posizione libreria)
(percorso tv tappeto)              (posizione finestra)
(percorso tappeto tv)              (posizione sedia)
(percorso tappeto sedia)
(percorso sedia tappeto)
(percorso porta tappeto)
(percorso tappeto porta)
(percorso finestra sedia)
(percorso sedia finestra)
(percorso finestra libreria)
(percorso libreria finestra)
```

Movimenti programmati

L'utente inserisce nel database del PROLOG le informazioni sull'ambiente in cui deve muoversi il robot. Il Prolog usa la procedura "vai(posto1,posto2)" per tracciare un sentiero da A a B (creando prima un percorso da A a C, poi quello da C a B).
Si noti l'impiego del simbolo di sottolineatura "_" nella clausola "traccia(A,A,_,R)". Questo può essere usato in PROLOG come "variabile anonima" o "jolly" e non viene inizializzato ad alcun valore quando la clausola viene eseguita. Qui la variabile anonima fa parte di una clausola secondo la quale esiste sempre un sentiero per spostarsi da A ad A

"Nuovo Mondo" III

La terza ed ultima parte del listato completo e suggerimenti per i possessori di Amstrad

Il programma è stato scritto appositamente per il Commodore 64 ma può essere eseguito (con qualche modifica) anche sui computer Amstrad.
In tal caso le istruzioni PRINT CHR$(147) vanno sostituite con CLS mentre tutte le linee che interrompono il programma in attesa che venga premuto un tasto, ad esempio:

<n.linea> GET I$: IF I$ = "" THEN <n. linea>

vanno sostituite con:

<n.linea> I$ = "": WHILE I$ = "": I$ = INKEY$: WEND

Infine occorre introdurre la linea: 5 MODE 1 per selezionare lo schermo a 40 colonne.

```
6530 REM SCIALUPPA DI SALVATAGGIO
6535 IF M(1)=1 THEN RETURN
6536 PRINT CHR$(147)
6540 M(1)=1
6550 S$="ALL'ORIZZONTE COMPARE UNA*": GOSUB 9100
6552 S$="SCIALUPPA DI SALVATAGGIO*": GOSUB 9100
6554 PRINT: GOSUB 9200
6556 S$="VEDI COL CANNOCCHIALE*": GOSUB 9100
6558 S$="CHE A BORDO CI SONO*": GOSUB 9100
6560 PRINT: GOSUB 9200
6562 S$="QUATTRO NAUFRAGHI*": GOSUB 9100
6563 GOSUB 9200
6564 S$="E UNA GROSSA CASSA!*": GOSUB 9100
6566 PRINT: GOSUB 9200
6568 S$="SE CAMBI ROTTA PER*": GOSUB 9100
6570 S$="SOCCORRERLI, IL VIAGGIO SI*": GOSUB 9100
6572 S$=" ALLUNGA DI DUE GIORNI!*": GOSUB 9100
6574 PRINT: GOSUB 9200
6576 S$="VUOI SALVARLI (S/N)?*": GOSUB 9100
6578 INPUT I$: I$=LEFT$(I$,1)
6580 IF I$<>"S" AND I$<>"N" THEN 6578
6585 IF I$="S" THEN 6600
6588 PRINT: GOSUB 9200
6590 S$="LA SCIALUPPA SCOMPARE...*": GOSUB 9100
6592 PRINT: GOSUB 9200
6594 S$=K$: GOSUB 9100
6596 GET I$: IF I$="" THEN 6596
6599 RETURN
6600 PRINT: GOSUB 9200
6610 EW=EW+2/7
6625 IF CN<>16 THEN 6630
6627 S$="NON PUOI PRENDERLI A*": GOSUB 9100
6628 S$="BORDO: NON C'E' SPAZIO.*":  GOSUB 9100
6629 GOTO 6592
6630 X=16-CN: IF X>3 THEN6635
6632 S$="C'E' POSTO SOLO PER*": GOSUB 9100
6633 PRINT X;" UOMINI"
6634 PRINT: GOSUB 9200
6635 S$="NAUFRAGHI SALVATI:*": GOSUB 9100
6638 X=0
6640 FOR T=1 TO 16
6645 IF TS(T,1)<>0 THEN 6679
6650 X=X+1
6655 IF X>4 THEN T=16: GOTO 6679
6660 CN=CN+1
6665 TS(T,1)=INT(RND(1)*5)+1
6668 TS(T,2)=INT(RND(1)*50)+50
6670 PRINT"1 ";C$(TS(T,1))
6679 NEXT
6680 PRINT: GOSUB 9200
6682 S$="LA CASSA CONTIENE:*": GOSUB 9100
6685 FOR T=1 TO 4
6690 X=INT(RND(1)*10)+10
6692 PRINT X;U$(T);" DI ";P$(T)
6693 IF PA(T)=-999 THEN PA(T)=0
6694 PA(T)=PA(T)+X
6695 NEXT
6699 GOTO 6592
```



L'equipaggio può essere colpito da un'epidemia...

```
6700 REM EPIDEMIA MORTALE
6705 IF M(2)=1 THEN RETURN
6706 PRINT CHR$(147)
6710 M(2)=1
6712 S$="EPIDEMIA TRA L'EQUIPAGGIO!*": GOSUB 9100
6714 PRINT: GOSUB 9200
6716 X=1
6718 FOR T=1 TO 16
6720 IF TS(T,1)=2 AND TS(T,2)<>0 AND TS(T,2)<>-999
     THEN X=0: T=16
6722 NEXT
6724 Y=1
6726 IF OA(1)<>0 AND OA(1)<>-999 THEN Y=0
6730 Z=((X+Y)*10)+5
```

...e la loro resistenza all'infezione dipende dalla presenza a bordo di un medico e di medicinali acquistati prima di salpare

```
6732 I$="A BORDO CI SONO ": IF X=0 AND Y=0 THEN6740
6734 IF X=1 THEN S$="MA NESSUN MEDICO*": GOSUB 9100:
     I$=" E "
6736 IF Y=1 THEN S$= I$+"NESSUNA MEDICINA*": GOSUB 9100
6740 S$="MOLTI UOMINI SI AMMALANO"*: GOSUB 9100
6745 PRINT: GOSUB 9200
6750 X=0
6755 FOR T=1 TO 16
6756 IF RND(1)<.3 THEN 6775
6760 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 6775
6765 TS(T,2)=TS(T,2)-Z
6770 IF TS(T,2)<1 THEN TS(T,2)=-999: X=X+1
6775 NEXT
6776 IF Y=1 THEN 6780
6777 S$="META' DELLE MEDICINE ESAURITA*": GOSUB 9100
6778 OA(1)=INT((OA(1)/2)+.5)
6780 PRINT: GOSUB 9200
6785 IF X=0 THEN 6797
6790 PRINT "E";X;
6792 S$="MEMBRI MORTI*"
6794 IF X=1 THEN S$="MEMBRO MORTO*"
6795 GOSUB 9100
6796 PRINT: GOSUB 9200
6797 S$=K$: GOSUB 9100
6798 GET I$: IF I$="" THEN 6798
6799 RETURN
```

I pirati possono attaccare la nave...

```
6800 REM PIRATI
5805 IF M(3)=1 THEN RETURN
6810 X=0
6812 FOR T=1 TO 16
6814 IF TS(T,2)=0 OR TS(T,2)=-999 THEN X=X+1
6815 NEXT
6816 IF X=16 THEN RETURN
6818 M(3)=1
6820 PRINT CHR$(147)
```

...a bordo ci sono fucili per difendersi e limitare le perdite?

```
6822 S$="  I PIRATI ATTACCANO LA NAVE!*": GOSUB 9100
6824 PRINT: GOSUB 9200
6825 K=2
6826 IF OA(2)=0 OR OA(2)=-999 THEN K=4
6828 S$=" NONOSTANTE LE ARMI*"
6830 IF K=4 THEN S$="NON CI SONO ARMI*"
6832 GOSUB 9200
6835 X=0
6836 FOR T=1 TO 16
6838 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 6845
```

```
6840 X=X+1: TS(T,2)=-999
6842 IF X=K THEN T=16
6845 NEXT
6850 PRINT X;
6855 S$="MEMBRO RIMANE UCCISO"
6856 IF X>1 THEN S$="MEMBRI RIMANGONO UCCISI"
6860 GOSUB 9100
6865 PRINT: GOSUB 9200
6890 S$=K$: GOSUB 9100
6895 GET I$: IF I$="" THEN 6895
6899 RETURN
6900 REM GUAI AL TIMONE!
6905 IF M(4)=1 THEN RETURN
6910 PRINT CHR$(147)
6915 M(4)=1
6920 S$="PROBLEMI AL TIMONE!": GOSUB 9100
6925 PRINT: GOSUB 9200
6928 X=4
6930 FOR T=1 TO 16
6935 IF TS(T,1)=3 AND TS(T,2)<>0 AND TS(T,2)<>-999
     THEN X=1: T=16
6938 NEXT
6940 S$="ANCHE SE C'E' IL MECCANICO"
6945 IF X=4 THEN S$="NON C'E' IL MECCANICO E"
6950 GOSUB 9100
6955 S$="IL VIAGGIO SI ALLUNGA DI": GOSUB 9100
6960 PRINT X; "SETTIMANE"
6965 EW=EW+X
6967 PRINT: GOSUB 9200
6969 S$=K$: GOSUB 9100
6970 GET I$: IF I$="" THEN 6970
6975 RETURN
7000 REM TEMPESTA
7005 IF M(5)=1 THEN RETURN
7010 PRINT CHR$(147)
7015 M(5)=1
7020 S$="UNA TEMPESTA DEVIA": GOSUB 9100
7022 S$="LA ROTTA DELLA NAVE": GOSUB 9100
7025 PRINT: GOSUB 9200
7028 X=2
7030 FOR T0=1 TO 16
7035 IF TS(T,1)=4 AND TS(T,2)<>0 AND TS(T,2)<>-999
     THEN X=1: T=16
7038 NEXT
7040 S$="ANCHE SE C'E' IL NAVIGATORE"
7045 IF X=2 THEN S$="NON C'E' IL NAVIGATORE E"
7049 GOTO 6950
```

Se le provviste stanno terminando, il giocatore può decidere di puntare verso un'isola per fare rifornimento ma, così facendo, la durata del viaggio si allunga

```
7050 REM ISOLA IN VISTA
7055 IF M(6)=1 THEN RETURN
7060 PRINT CHR$(147)
7065 M(6)=1
7070 S$="LE CARTE MOSTRANO LA ": GOSUB 9100
7071 S$="PRESENZA DI UN'ISOLA": GOSUB 9100
7072 S$="SULLA QUALE RIFORNIRSI DI": GOSUB 9100
7073 S$="SCORTE. APPRODARE PERO'": GOSUB 9100
7074 S$="FA PERDERE TEMPO": GOSUB 9100
7075 PRINT: GOSUB 9200
7080 S$="DESIDERI APPRODARE (S/N)?": GOSUB 9100
7082 INPUT I$: I$=LEFT$(I$,1)
7084 IF I$<>"S" AND I$<>"N" THEN 7082
7086 PRINT: GOSUB 9200
7090 IF I$="N" THEN 7145
7100 S$="L'ISOLA E' RAGGIUNTA": GOSUB 9100
7105 S$="SCORTE RACCOLTE:" GOSUB 9100
7106 IF B$="N" THEN 7110
7107 PRINT: GOSUB 9200
7108 PRINT "NESSUNA!!!": GOSUB 9200
7109 S$="(RICORDA L'ALBATRO!)": GOSUB 9100: GOTO 7130
7110 FOR T=1 TO 4
7112 IF RND(1)<.25 THEN 7129
7115 X=INT(RND(1)*10)+5
7120 PRINT X; U$(T);"  DI"; P$(T)
7122 IF PA(T)=-999 THEN PA(T)=0
7125 PA(T)=PA(T)+X
7129 NEXT
7130 S$="ADESSO IL VIAGGIO RICHIEDE": GOSUB 9100
7135 X=INT(RND(1)*2)+1
7139 PRINT X;: S$="SETTIMANE IN PIU'": GOSUB 9100
7140 EW=EW+X
7145 PRINT: GOSUB 9200
7150 S$=K$: GOSUB 9100
7155 GET I$: IF I$="" THEN 7155
7159 RETURN
```

La coincidenza di sette circostanze diverse può scatenare l'ammutinamento dell'equipaggio. Questa routine crea un fattore d'ammutinamento, MF, verificando le condizioni a bordo alla fine di ogni settimana di viaggio, e sommando il risultato ottenuto a MF

```
7200 REM AMMUTINAMENTO
7210 MF=0
7215 IF H$="S" THEN MF=MF+30
7220 NC=0
7225 FOR T=1 TO 16
7228 IF TS(T,1)=5 AND TS(T,2)<>0 AND TS(T,2)<>-999 THEN
     NC=1: T=16
7230 NEXT
7235 IF NC=0 THEN MF=MF+30
7240 IF A$="S" THEN MF=MF-20
7245 IF B$="S" THEN MF=MF+30
7250 IF CN>12 THEN MF=MF+30
7255 IF WT>MO THEN MF=MF+30
7260 IF WK>8 THEN MF=MF+((WK-8)*10)
7275 MF=MF+INT(RND(1)*30)
```

Se il fattore d'ammutinamento supera 75 (ma è minore di 100), il programma avverte il giocatore che l'equipaggio progetta un ammutinamento

```
7280 IF MF<75 THEN RETURN
7282 PRINT CHR$(147)
7284 IF MF>100 THEN 7300
7285 S$="LE CONDIZIONI DI VITA A": GOSUB 9100
7286 S$=" BORDO PEGGIORANO": GOSUB 9100
7287 S$=" E ALCUNI MARINAI": GOSUB 9100
7288 S$=" TRAMANO UN AMMUTINAMENTO": GOSUB 9100
7290 PRINT: GOSUB 9200
7292 S$=K$: GOSUB 9100
7294 GET I$: IF I$="" THEN 7294
7299 RETURN
```

Se il fattore d'ammutinamento supera il valore 100, l'equipaggio si ribella e, dopo aver comunicato al capitano le proprie lamentele, lo abbandona in mare aperto

```
7300 PRINT CHR$(147)
7305 PRINT: GOSUB 9200
7310 S$="L'EQUIPAGGIO SI E' AMMUTINATO": GOSUB 9100
7312 S$=" PERCHE':": GOSUB 9100
7313 X=0
7314 IF H$<>"S" THEN 7320
7315 GOSUB 9200: X=X+1: PRINT X;
7316 S$="LE RAZIONI SONO DIMEZZATE": GOSUB 9100
7318 S$=" PER PARTE DEL VIAGGIO": GOSUB 9100
7320 IF NC<>0 THEN 7325
7321 GOSUB 9200: X=X+1: PRINT X;
7322 S$="MANCA IL CUOCO": GOSUB 9100
7324 S$=" E IL CIBO E' IMMANGIABILE": GOSUB 9100
7325 IF B$<>"S" THEN 7330
7326 GOSUB 9200: X=X+1: PRINT X;
7327 S$="L'ALBATRO E' STATO UCCISO!": GOSUB 9100
7330 IF CN<13 THEN 7335
7331 GOSUB 9200: X=X+1: PRINT X;
7332 S$="LA NAVE E' SOVRAFFOLLATA": GOSUB 9100
7335 IF MO>=WT THEN 7340
7336 GOSUB 9200: X=X+1: PRINT X;
7337 S$="MANCA IL DENARO": GOSUB 9100
7338 S$=" PER LE PAGHE": GOSUB 9100
7340 IF WK<=8 THEN 7350
7341 GOSUB 9200: X=X+1: PRINT X;
7342 S$="IL VIAGGIO DURA DA ": GOSUB 9100
7343 S$="PIU' DI 8 SETTIMANE": GOSUB 9100
7350 PRINT: GOSUB 9200
7360 S$="GLI AMMUTINATI PRENDONO ": GOSUB 9100
7362 GOSUB 9200
7363 S$="LA NAVE E FUGGONO": GOSUB 9100
7370 GOSUB 9200
7372 S$="ABBANDONANDO IL CAPITANO ": GOSUB 9100
7373 S$="IN MARE APERTO": GOSUB 9100
7374 S$=" BUON BAGNO!!!!": GOSUB 9100
7375 PRINT: GOSUB 9200
7380 PRINT"          ,FINE DEL GIOCO"
7382 END
```

Il programma usa ampiamente queste brevi subroutine per produrre effetti particolari, come il ritardo nella visualizzazione dei messaggi

```
9100 REM VISUALIZZAZIONE RALLENTATA
9110 FOR S3=1 TO 32
9115 IF MID$(S$,S3,1)="" THEN S3=32: GOTO 9140
9120 PRINT MID$(S$,S3,1);
9130 FOR S4=1 TO 20: NEXT
9140 NEXT: PRINT
9199 RETURN
9200 REM CICLO DI RITARDO LUNGO
9210 FOR S5=1 TO 1000: NEXT
9299 RETURN
9300 REM RIDUCE DI WF IL RENDIMENTO DELL'EQUIPAGGIO
9310 FOR S1=1 TO 16
9315 IF TS(S1,2)=0 THEN 9340
9320 TS(S1,2)=TS(S1,2)-WF
9330 IF TS(S1,2)<1 THEN TS(S1,2)=-999
9340 NEXT
9399 RETURN
```



La nave arriva a destinazione...

```
10000 REM APPRODO AL NUOVO MONDO
10001 PRINT CHR$(147): GOSUB 9200
10005 S$="SEI GIUNTO NEL NUOVO MONDO•": GOSUB 9100
10006 PRINT: GOSUB 9200
10007 S$="MENTRE LA NAVE SI AVVICINA•": GOSUB 9100
10008 S$=" VENGONO INCONTRO ALCUNI•":GOSUB 9100
10009 S$=" INDIGENI •":GOSUB 9100
10010 PRINT: GOSUB 9200
10015 IF OA(2)=0 THEN 10050
10017 S$="SONO ARMATI E BELLICOSI!!•": GOSUB 9100
10018 PRINT: GOSUB 9200
10020 S$="APRI IL FUOCO (S/N)?•": GOSUB 9100
10022 INPUT I$: I$=LEFT$(I$,1)
10024 IF I$<>"N" AND I$<>"S" THEN 10022
10026 IF I$="N" THEN 10050
10028 PRINT: GOSUB 9200
10030 S$="MOLTI INDIGENI RIMANGONO UCCISI•": GOSUB 9100
10032 S$=" MA DURANTE LA NOTTE•": GOSUB 9100
10034 S$=" NE ARRIVANO ALTRI•": GOSUB 9100
10036 S$=" CHE BRUCIANO LA TUA NAVE!!!!•": GOSUB 9100
10038 PRINT: PRINT: GOSUB 9200
10040 S$="          GAME OVER•": GOSUB 9100
10042 END
10044 GOTO 10042
10050 S$="TI CONDUCONO A CONOSCERE IL•": GOSUB 9100
10052 S$=" LORO CAPO, CHE HA GIA'": GOSUB 9100
10053 S$=" COMMERCIATO CON L'EUROPEI.•":GOSUB 9100
10054 S$=" E' MOLTO AMICHEVOLE.•": GOSUB 9100
10056 GOSUB 9200
10058 S$="L'EQUIPAGGIO VIENE SFAMATO•":GOSUB 9100
10059 S$="E SI RIPOSA.•":GOSUB 9100
10060 PRINT: GOSUB 9200
10062 S$="IL GIORNO DOPO INIZIANO•":GOSUB 9100
10063 S$="LE TRATTATIVE.•":GOSUB 9100
10064 PRINT: GOSUB 9200
10066 S$=K$: GOSUB 9100
10068 GET I$: IF I$="" THEN 10068
10069 RETURN
```

...e iniziano i baratti

```
10070 PRINTCHR$(147): GOSUB 9200: REM BARATTO
10072 IF OA(2)=0 THEN 10080
10074 S$="IL CAPO RIFIUTA I FUCILI:•": GOSUB 9100
10076 S$="TURBANO LA SUA GENTE!•": GOSUB 9100
10078 PRINT: GOSUB 9200
10080 IF OA(3)<>0 OR OA(4)<>0 OR OA(5)<>0 OR OA(6)<>0
      THEN 10100
10085 S$="UN INCENDIO HA DISTRUTTO•": GOSUB 9100
10090 S$="TUTTE LE MERCI DA BARATTARE•": GOSUB 9110
10095 GOTO 10038
10100 S$="IL CAPO INDIGENO TI OFFRE•": GOSUB 9100
10102 S$="PERLE, MANUFATTI E SPEZIE•": GOSUB 9100
10104 S$="IN CAMBIO DI COLTELLI,•": GOSUB 9100
10106 S$="SALE, STOFFE E MONILI•": GOSUB 9100
10108 PRINT: GOSUB 9200
10110 S$="IL VALORE INIZIALE DELLE•": GOSUB 9100
10112 S$="MERCI OFFERTE ERA:•": GOSUB 9100
10114 S$="1 PERLA = 2 MONETE D'ORO•": GOSUB 9100
10116 S$="1 MANUFATTO = 2 MONETE D'ORO•": GOSUB 9100
10118 S$="1 GR. SPEZIE = 1 MONETA D'ORO•": GOSUB 9100
10120 PRINT: GOSUB 9200
10122 S$="MA, RIMPATRIANDO, PUO'•": GOSUB 9100
10124 S$="RISULTARE DIVERSO.•": GOSUB 9100
10125 PRINT: GOSUB 9200: S$=K$: GOSUB 9100
10126 GET I$: IF I$="" THEN 10126
10130 FOR T=3 TO 6
10135 IF OA(T)=0 THEN 10200
10140 PRINT CHR$(147): GOSUB 9200
10145 PRINT "ATTUALMENTE POSSIEDI";OA(T);
10150 IF T=3 THEN S$="SACCHI DI SALE•"
10151 IF T=4 THEN S$="BALLE DI STOFFA•"
10152 IF T=5 THEN S$="COLTELLI•"
10153 IF T=6 THEN S$="MONILI•"
10155 GOSUB 9100
10156 PRINT: GOSUB 9200
10160 S$="IL CAPO TI OFFRE:•": GOSUB 9100
10165 PRINT "          ";OA(T)*EQ(T-2,1);" PERLE"
10166 PRINT "      O ";OA(T)*EQ(T-2,2);" MANUFATTI"
10167 PRINT "OPPURE ";OA(T)*EQ(T-2,3);" GRAMMI DI
      SPEZIE•"
10168 PRINT: GOSUB 9200
10170 S$="DESIDERI PERLE, MANUFATTI•": GOSUB 9100
10172 S$="O SPEZIE ?•": GOSUB 9100
10174 S$="(PREMERE 1, 2 O 3)•": GOSUB 9100
10175 INPUT I$
10176 I=VAL(I$): IF I<1 OR I>3 THEN 10174
10180 AO(I)=AO(I)+(OA(T)*EQ(T-2,I))
10190 PRINT: PRINT "CARICO A BORDO DI ";T$(I);
      " EFFETTUATO."
10192 S$=K$: GOSUB 9100
10194 GET I$: IF I$="" THEN 10194
10200 NEXT
10210 PRINT: PRINT: GOSUB 9200
10215 S$="SCAMBIO DI MERCI TERMINATO•": GOSUB 9100
10216 PRINT: GOSUB 9200
10218 S$="ADESSO POSSIEDI:•": GOSUB 9100
10220 PRINT AO(1);" PERLE"
10222 PRINT AO(2);" MANUFATTI"
10224 PRINT AO(3);" GRAMMI DI SPEZIE"
10226 PRINT: GOSUB 9200
10228 S$=K$: GOSUB 9100
10229 GET I$: IF I$="" THEN 10229
10230 RETURN
```



Il capitano viene coinvolto in un tentativo di rivolta? I rischi sono elevati ma non inferiori a quelli corsi se la rivolta fallisce

```
10300 REM RIVOLTA
10305 IF OA(2)=0 THEN RETURN
10310 PRINT CHR$(147):GOSUB 9200
10315 S$="DURANTE LA NOTTE UN RIVALE DEL•": GOSUB 9100
10316 S$="CAPO VISITA IN SEGRETO LA NAVE•": GOSUB 9100
10317 PRINT:GOSUB 9200
10318 S$="DESIDERA ACQUISTARE ARMI PER•": GOSUB 9100
10320 S$="UNA RIVOLTA•": GOSUB 9100
10322 PRINT:  GOSUB 9200
10324 S$="OFFRE 30 PERLE PER OGNI ARMA•": GOSUB 9100
10326 S$="VUOI VENDERGLI ARMI (S/N)?•": GOSUB 9100
10328 INPUT I$:I$=LEFT$(I$,1)
10330 IF I$<>"N" AND I$<>"S" THEN 10328
10332 IFI$="S" THEN 10400
10334 PRINT: GOSUB 9200
10336 S$="IL CAPO HA SAPUTO ED E'•": GOSUB 9100
10338 S$="RICONOSCENTE: TI REGALA DELLE•": GOSUB 9100
10340 S$="UTILI SCORTE PER IL •": GOSUB 9100
10342 S$="VIAGGIO DI RITORNO•": GOSUB 9100
10344 GOSUB 9200
10345 IF RND(1)<.75 THEN 10350
10346 S$="E ANCHE 50 PERLE!!•": GOSUB 9100
10348 AO(1)=AO(1)+50
10350 PRINT: GOSUB 9200
10352 S$=K$: GOSUB 9100
10354 GETI$: IFI$=""THEN10354
10359 RETURN
10400 PRINTCHR$(147): GOSUB 9200
10405 IFRND(1)<.75 THEN 10450
10410 S$="LA RIVOLTA HA SUCCESSO!•": GOSUB 9100
10412 PRINT: GOSUB 9200
10415 S$="IL NUOVO CAPO TI REGALA•": GOSUB 9100
10420 S$="UTILI SCORTE PER IL VIAGGIO•": GOSUB 9100
10425 S$="DI RITORNO.•":GOSUB 9100
10429 AO(1)=AO(1)+(OA(2)•30): REM AGGIUNGE LE PERLE
10430 OA(2)=0
10431 GOTO10350
10450 S$="LA RIVOLTA FALLISCE!!•": GOSUB 9100
10452 PRINT: GOSUB 9200
10455 S$="IL VECCHIO CAPO E' ADIRATO:•": GOSUB 9100
10457 S$="FA BRUCIARE LA TUA NAVE E•": GOSUB 9100
10458 S$="SACCHEGGIA TUTTO!!•": GOSUB 9100
10459 PRINT: GOSUB 9200
10460 S$=" FINE DEL GIOCO!!•": GOSUB 9100
10462 END
10464 GOTO10462
```

Infine la nave torna al porto di partenza e il programma valuta le attitudini commerciali del giocatore

```
10500 REM CONCLUSIONE DEL VIAGGIO
10501 PRINTCHR$(147): GOSUB 9200
10505 S$="CON L'EQUIPAGGIO IN FORMA E•": GOSUB 9100
10507 S$="VENTI FAVOREVOLI IL RITORNO•": GOSUB 9100
10508 S$="DURA SOLTANTO 8 SETTIMANE.•": GOSUB 9100
10512 WW=0
10514 FOR T=1 TO 5
10516 WW=WW+(B•CC(T)•WG(T))
10518 NEXT
10519 PRINT: GOSUB 9200
10520 S$="SALARI PER IL RITORNO = •": GOSUB 9100
10522 PRINTWW;"MONETE D'ORO"
10524 PRINT: GOSUB 9200
10526 S$="ALL'ARRIVO IN PORTO•":GOSUB 9100
10528 S$="VENDI LE MERCI IL CUI VALORE•": GOSUB 9100
10530 S$="ATTUALE E':•": GOSUB 9100
10532 PRINT"PERLE - ";V2(1);"MONETE D'ORO"
10534 PRINT"MANUFATTI - ";V2(2);"MONETE D'ORO"
10536 PRINT"SPEZIE - ";V2(3);"MONETE D'ORO"
10538 PRINT:S$="PER UN TOTALE DI •": GOSUB 9100
10540 X=(AO(1)•V2(1))+(AO(2)•V2(2))+(AO(3)•V2(3))
10542 PRINTX;"MONETE D'ORO"
10545 PRINT:S$=K$: GOSUB 9100: PRINT
10547 GET I$:IF I$="" THEN 10547
10550 S$="ADESSO POSSIEDI:•": GOSUB 9100
10552 PRINTMO+X;"MONETE D'ORO"
10555 PRINT: GOSUB 9200
10556 S$="SALARI PER IL RITORNO:•": GOSUB 9100
10577 PRINTWT+WW;"MONETE D'ORO"
10559 PRINT: GOSUB 9200
10560 S$="IL GIOCO TERMINA CON:•": GOSUB 9100
10562 Z=MO+X-WT-WW
10565 PRINTZ;"MONETE D'ORO"
10566 PRINT:GOSUB 9200
10567 PRINT:S$="SEI NOMINATO •": GOSUB 9100: PRINT
10568 IF Z>3200 THEN S$="GRAN FINANZIERE•":
      GOSUB 9100: END
10569 IF Z>2500 THEN S$="MAESTRO COMMERCIANTE•":
      GOSUB 9100: END
10570 IF Z>2000 THEN S$="MERCANTE DI TERZA CLASSE•":
      GOSUB 9100: END
10571 IF Z>1000 THEN S$="... UN VERO POLLO!•":
      GOSUB 9100: END
10572 S$="DISASTRO DELL'ANNO!•"
10573 GOSUB 9100: END
```

Nastri e misure

Le routine ROM dello Spectrum per l'accesso ai dati su cassetta

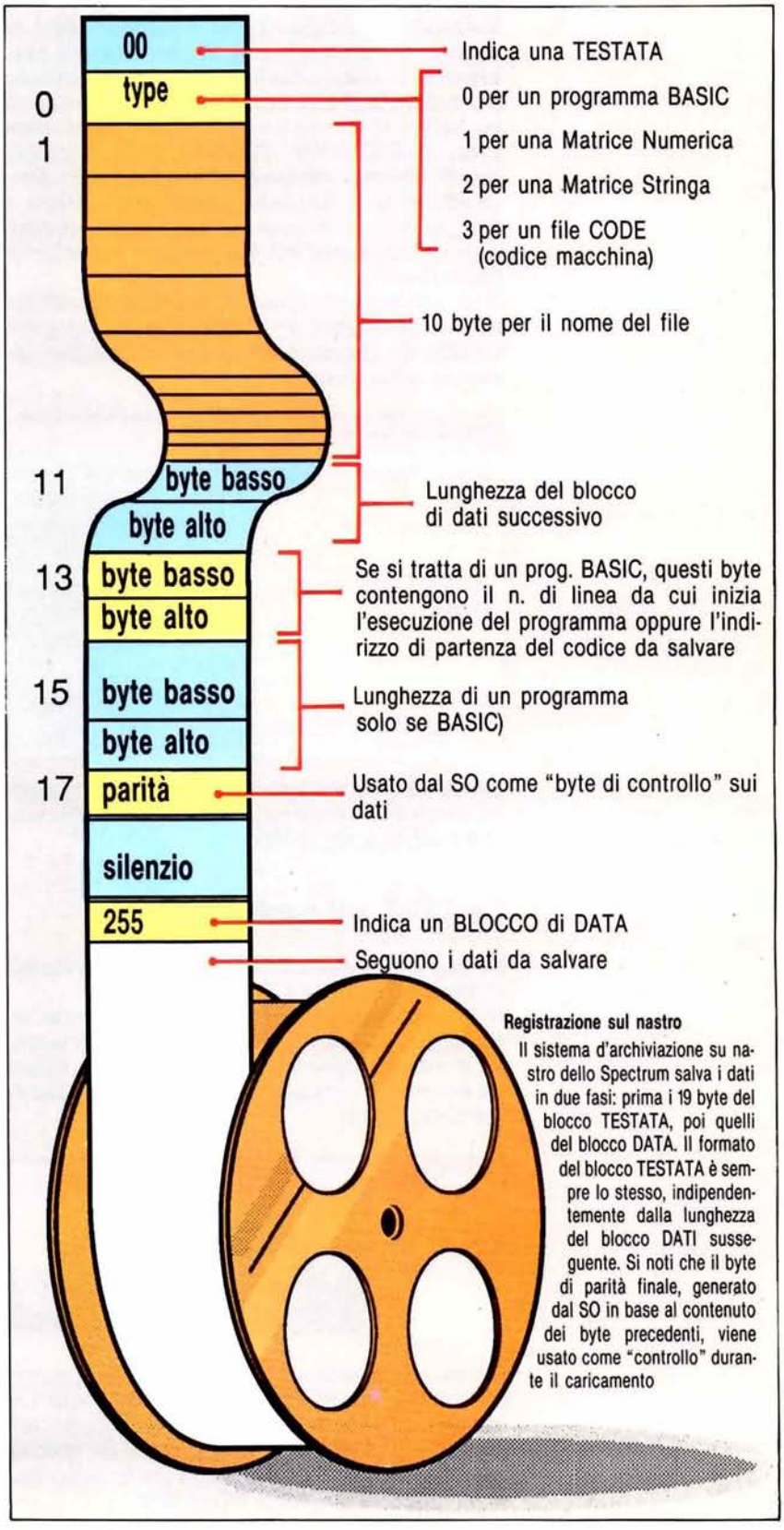
La versione base dello Spectrum Sinclair possiede solo un'interfaccia per il salvataggio e il caricamento di programmi e dati su nastro. L'innesto dell'Interface 1 permette di collegare l'apparecchio ai microdrive, a un'interfaccia seriale e alle "reti" locali. Queste unità possono essere considerate sistemi d'archiviazione alternativi paragonabili a quelli offerti, ad esempio, dal Micro BBC. Tuttavia, mentre sul Micro BBC il sistema d'archiviazione viene selezionato con un comando "stella" (*), lo Spectrum usa una diversa sintassi di comando per distinguere i comandi relativi al sistema d'archiviazione su nastro da quelli per gli altri sistemi disponibili. Per esempio, il comando:

```
SAVE "testi1" LINE 1
```

salva un programma BASIC sul nastro in modo che, quando lo si ricarica in memoria, la sua esecuzione inizi dalla linea 1. Per ottenere un identico salvataggio su microdrive, è necessario usare un comando molto più complesso:

```
SAVE *"m";1;"testi1" LINE 1
```

dove m indica il microdrive in uso, 1 è il codice numerico del drive selezionato e testi1 il nome del file da salvare. Il sistema di archiviazione su microdrive viene descritto più avanti. Esaminiamo adesso come viene usato il sistema di archiviazione su nastro per i programmi in codice macchina.
Indipendentemente dal loro tipo, i dati salvati su nastro vengono sempre organizzati nel modo indicato in figura. I primi 19 byte trasmessi costituiscono la "testata" e contengono informazioni relative al blocco di dati successivo. Il primo e l'ultimo byte della testata vengono forniti dalle routine del sistema operativo che registrano i dati sul nastro. Gli altri byte devono essere specificati prima della chiamata per la routine ROM che scrive i dati sul nastro. Il byte del "tipo" comunica al SO la natura dei dati contenuti nel blocco ed è seguito da un nome di file composto da 10 byte: ciò spiega il limite massimo di 10 caratteri per i nomi di file sullo Spectrum. Se il nome occupa meno byte, gli spazi vuoti vengono riempiti con codici ASCII di spaziatura (32).
I dati della testata successivi al decimo byte comunicano al SO altre informazioni, ad esempio sulle modalità di caricamento dei dati in memoria. L'ultimo byte della testata è usato per verificare se sono stati commessi errori durante la registrazione del nastro: in caso affermativo il SO genera l'appropriato messaggio d'errore.



Registrazione sul nastro
Il sistema d'archiviazione su nastro dello Spectrum salva i dati in due fasi: prima i 19 byte del blocco TESTATA, poi quelli del blocco DATA. Il formato del blocco TESTATA è sempre lo stesso, indipendentemente dalla lunghezza del blocco DATI susseguente. Si noti che il byte di parità finale, generato dal SO in base al contenuto dei byte precedenti, viene usato come "controllo" durante il caricamento

Il blocco principale dei dati inizia con un byte, fornito dalla routine ROM di "scrittura su nastro", che contiene il valore 255. Ad esso seguono quindi i dati, che terminano con un altro byte di verifica, generato anch'esso dalla citata routine. Sia nella testata sia nei blocchi dati, il valore reale registrato nell'ultimo byte del blocco dipende dai byte che lo seguono nella trasmissione.
Esaminiamo adesso come vengono usate le routine d'interfaccia per la cassetta dai programmi in codice macchina. Sullo Spectrum, le routine ROM che gestiscono le operazioni su nastro si trovano agli indirizzi ROM compresi tra $04C2 e $09F3. Si tratta di un programma di discrete dimensioni poiché, sullo Spectrum, tutte le funzioni di temporizzazione e di generazione di tono necessarie per caricare le informazioni sul nastro sono controllate dal software.
Non esistono variabili di sistema specifiche per le operazioni su nastro, ma la seguente tabella ne contiene alcune usate "saltuariamente" dalle routine.

Variabile di sistema	Descrizione
TADDR (alle locazioni 23668 e 23669)	Usata dalle routine di sistema per stabilire quali operazioni su nastro eseguire in base al comando BASIC ricevuto
BORDCR (alla locazione 23624)	Usata per ripristinare il colore del bordo (Border Colour) originale dopo un'operazione su nastro
XPTR (alle locazioni 23647 e 23648)	Usata per conservare temporaneamente il registro IX

Le variabili di sistema VARS, ELINE e PROG vengono usate per caricare in memoria le informazioni salvate sul nastro.

Le SAVE sul nastro

La routine responsabile di queste operazioni si trova all'indirizzo ROM $04C2.
Normalmente essa viene chiamata 2 volte: la prima per salvare le informazioni della testata, la seconda per registrare sul nastro i dati da salvare. La seguente tabella indica i parametri da usare:

REGISTRO	TESTATA	BLOCCO DI DATI
A	0	255
DE	17	Lunghezza dei dati
IX	Indirizzo iniziale della testata o del blocco dei dati	

Come si può notare, il registro IX viene usato per puntare al dato da salvare. Per chiarire il funzionamento della routine, si prenda ad esempio il seguente programma in codice macchina, che salva su nastro 100 byte di dati (a partire dall'indirizzo ROM 0000).

```
                ;questa routine salva su nastro 100 byte a partire dall'indirizzo 0000
                ;
                ;salva la testata
3E00            ld      a,0         ;indica un BLOCCO TESTATA
DDE5            push    ix          ;appoggia ix sullo stack
DD21D328        ld      ix, header  ;indirizzo del blocco di testata
111100          ld      de,17       ;n. dei byte della testata
CDC204          call    #04c2       ;scrive la testata sul nastro
                ;salva i dati
DD210000        ld      ix,0000     ;indirizzo del primo byte da salvare
116400          ld      de,100      ;n. di byte da salvare
3EFF            ld      a,255       ;indica un BLOCCO DATI
CDC204          call    #04c3       ;scrive i dati sul nastro
DDE1            pop     ix          ;ripristina il valore di IX
C9              ret                 ;torna al BASIC
03              header: defb    3           ;tipo di dato 3 = blocco di un file
54455354        defm    "PROGTEST   ;nome di file + spazi = 10 caratteri
64              defb    100         ;byte-basso della lunghezza dati
00              defb    0           ;byte-alto della lunghezza dati
00              defb    00          ;byte-basso dell'indirizzo iniziale
00              defb    00          ;byte-alto dell'indirizzo iniziale
00              defb    00          ;usato solo per il BASIC
00              defb    00          ;per il n. di linea di partenza
```

La routine salva l'area di memoria specificata ma non visualizza messaggi d'errore come fanno invece le routine di gestione del nastro chiamate dal BASIC. Tuttavia questa mancanza è irrilevante poiché i dati possono essere ricaricati per verificare il corretto funzionamento del programma, usando il comando:

```
LOAD "PROGTEST" CODE 40000
```

Adesso si confrontino i byte caricati con quelli posti alle locazioni ROM da 0 a 99 ricordando che, sebbene il blocco di dati della TESTATA inizi con il byte del "tipo", il primo byte effettivamente inviato al nastro viene fornito dal SO (0 per una testata e 225 per un blocco di dati).
Modificando nel modo seguente il blocco TESTATA del listato precedente è possibile salvare un programma BASIC dall'interno di un programma in codice macchina:

```
00              testata:    defb    0           ;0 = programma BASIC
73737373                    defm    "sssss      ;nome di file (10 caratteri)
                            sssss"
00                          defb    nn          ;byte-basso| lunghezza del
00                          defb    nn          ;byte-alto| prog. + variabili
00                          defb    nn          ;byte-basso| n. della linea
00                          defb    nn          ;byte-alto| di partenza
00                          defb    nn          ;byte-basso| lunghezza del
00                          defb    nn          ;byte-alto| solo programma
```

La lunghezza complessiva del programma e delle variabili può essere calcolata sottraendo il valore di PROG da ELIN, quella del solo programma sottraendo il valore di PROG da VARS.
Per evitare che il programma caricato venga eseguito automaticamente, occorre assegnare l'indirizzo 32768 al parametro numero di linea di esecuzione del programma, previsto dal listato. Analogamente è possibile simulare in codice macchina il comando SAVE SCREEN$ del BASIC, assegnando le locazioni $4000 e $1B00 rispettivamente all'indirizzo iniziale e alla lunghezza del programma da salvare. Infine è possibile perfezionare la routine di salvataggio su nastro ricorrendo alle tecniche descritte nelle precedenti parti del corso. Ad esempio, si può introdurre un messaggio guida (prompt) sullo schermo e una semplice routine che rimane in attesa finché non viene premuto un tasto.

Caricamento dal nastro

Anche la routine ROM che carica i dati dal nastro viene chiamata due volte: la prima per leggere la TESTATA, la seconda per il blocco dei dati. La routine si trova all'indirizzo ROM &0556; la seguente tabella indica i parametri necessari (assieme a quelli per il codice di verifica):

Registro	LOAD	Verifica
Flag carry	1	0
A	0 per la testata, 255 per il blocco dei dati	
IX	Punta alla locaz. di memoria nella quale devono essere caricati i byte	
DE	Numero di byte da caricare; D deve contenere un valore compreso tra 0 e 254	

Anche questa routine occupa una discreta area di lavoro, circa 34 byte o più, per alloggiare due blocchi TESTATA. Per caricare un file in RAM a partire da un indirizzo dato, è necessario costruire un secondo blocco TESTATA contenente le informazioni relative al file da caricare e confrontare i dati TESTATA del file sul nastro con la TESTATA creata in memoria per localizzare il file in questione.
Quando viene chiamata la routine, lo stato del Flag di carry deve essere quello indicato nella tabella. Inoltre la parte del codice sottoposta a verifica deve trovarsi all'indirizzo puntato dal registro IX. Al termine della routine ROM per il "caricamento dal nastro", lo stato del flag C indica il risultato dell'operazione: se contiene il valore 1, il caricamento è stato eseguito correttamente. Se invece, durante il tentativo di caricamento del blocco TESTATA, la routine incontra per primo il blocco dei dati, allora pone il Flag di carry a 0 (tutti gli errori di caricamento dal nastro vengono gestiti dal SO dello Spectrum). Esaminiamo adesso una semplice routine dimostrativa, che carica una TESTATA dal nastro in RAM.

```
                ;carica una testata dal nastro in RAM
                ;
37              scf             ;pone il flag carry a 1 = LOAD
3E00            ld    a,0       ;0 indica una testata
DDE5            push  ix        ;appoggia ix sullo stack
DD2148EE        ld    ix,61000  ;carica la testata (da 61000)
111100          ld    de,17     ;n. dei byte in testata
CD5605          call  #0556     ;esegue il caricamento
DDE1            pop   ix        ;ripristina ix
C9              ret
```

Caricata la TESTATA, si può esaminarne il contenuto confrontando il nome del file con quello specificato, accertando che il file sia del tipo richiesto ed, eventualmente, verificando la lunghezza del blocco dei dati. Il seguente programma verifica solo il nome e, se questo corrisponde, carica in memoria il blocco dei dati che segue la TESTATA a partire da uno specifico indirizzo.

```
                ;localizza e carica PROGTEST
                ;
DDE5                    push  ix        ;appoggia IX sullo stack
37              ciclo:  scf             ;pone il flag C a 1 = LOAD
3E00                    ld    a,0       ;0 = file BASIC
111100                  ld    de,17     ;n. di byte in testata
DD21CA2B                ld    ix,head2  ;carica testata in testa2
CD5605                  call  #0556
D27B2B                  jp    nc,loop   ;ripete se non trova la testata
060A                    ld    b,10      ;n. di byte nel nome di file
11BA2B                  ld    de,head+1 ;punta al nome di file richiesto
21CB2B                  ld    hl,head2+1 ;punta HL al nome di file trovato
1A              nome:   ld    a,(de)    ;riceve in A un car. della testata
BE                      cp    (hl)      ;lo confronta con la testata trovata
2007                    jr    nz,no     ;car. diversi: esce dal ciclo
13                      inc   de        ;riceve il carattere seguente
23                      inc   hl
05                      dec   b         ;decrementa il contatore
20F7                    jr    nz,nome   ;car. uguali: confronta il car. seguente
1802                    jr    ok        ;tutti i car. corrispondono
18DB            no:     jr    loop      ;verifica fallita, prova di nuovo
                ;trova la testata giusta e prepara il caricamento
DD21B92B        ok:     ld    ix,head   ;riceve l'indirizzo in IX
DD6E0D                  ld    l,(ix+13) ;lo usa per caricare
DD660E                  ld    h,(ix+14) ;i dati in memoria
E5                      push  hl        ;appoggia l'indirizzo sullo stack
DDE1                    pop   ix        ;e lo inserisce in IX
3EFF                    ld    a,255     ;indica un caricamento di dati
                ;l'istr. successiva chiede all'utente d'inserire la lunghezza dei dati
11AF2B                  ld    de,nn     ;inserisce la lunghezza dei dati
37                      scf             ;segnala un caricamento (LOAD)
CD5605                  call  #0556     ;lo esegue
DDE1                    pop   ix        ;ripristina ix
C9                      ret
                ;seguono gli attributi del file richiesto
03              testa:  defn  3         ;indica un blocco CODICE
54455354                defm  "PROGTEST" ;nome del file
00                      defb  0         ;byte-basso e byte-alto della
00                      defb  0         ;lunghezza dei dati usati per
48                      defb  72        ;il caricamento
EE                      defb  238       ;indirizzo usato: 61000
00                      defb  0         ;usato solo per
00                      defb  0         ;prog. BASIC
                ;spazio per la testata caricata per la verifica
                testa2: defs  17
```

Anche in questo caso non vengono visualizzati messaggi durante l'operazione. La routine carica un bocco di dati chiamato PROGTEST nell'indirizzo specificato nell'area di memoria TESTA e può essere usata anche per caricare file con un nome diverso. Inoltre a essa può essere aggiunta una procedura che verifica se il tipo del file è quello richiesto. Si noti che le routine ROM, come le funzioni SAVE e LOAD del BASIC, possono essere interrotte premendo il tasto Break.
La descrizione delle operazioni di salvataggio e caricamento del SO dello Spectrum si conclude con un breve programma che carica dal nastro le testate dei file e ne visualizza alcune caratteristiche come la lunghezza del file, l'indirizzo di partenza e così via. La routine di codice macchina immagazzinata nella linea d'istruzione DATA carica un blocco TESTATA e usa il BASIC per esaminarne il contenuto.

```
 5 CLEAR 59999
10 FOR I=0 TO 19
20 READ A:POKE (60000+I),A
30 NEXT I
40 RANDOMIZE USR 60000
50 LET tipo=PEEK 60020
60 LET lung=PEEK 60031+256*PEEK 60032
70 LET inizio=PEEK 60033+256*PEEK 60034
80 LET L$=""
90 FOR I=60021 TO 60030: LET L$=L$+CHR$(PEEK I): NEXT I
100 PRINT "Nome: ";L$
110 IF tipo=0 THEN LET f$="BASIC"
120 IF tipo=1 THEN LET f$="Matrice Numerica"
130 IF tipo=2 THEN LET f$="Matrice Stringa"
140 IF tipo=3 THEN LET f$="Codice macchina"
150 PRINT "Tipo: ";f$
160 IF tipo=0 THEN PRINT "N. di linea per Autoesec.:";inizio
170 IF tipo<>0 THEN PRINT "Indirizzo di partenza: ";inizio
180 PRINT "Lunghezza: ";lung
190 GOTO 40
200 DATA 221,229,62,0,55,221,33,116,234,17,17,0,205,86,5,48,241,221,225,201
```

Football Manager

Alcuni giochi non richiedono solo prontezza di riflessi e una mano ferma

Lanciato sul mercato agli inizi del 1980, il *Football Manager*, ideato dalla Addictive Games Limited, ha mantenuto a lungo un buon livello di vendite. In un settore dove i prodotti hanno generalmente una "vita" molto breve, indipendentemente dal successo ottenuto al momento del lancio, questo programma si distingue nettamente da tutti gli altri.

Ciò può stupire, se si considera che, in fondo, il programma non presenta aspetti particolarmente innovativi: i principi sui quali è basato sono in effetti molto semplici. Il ruolo del giocatore è quello del "manager" di una squadra di calcio che deve prendere le giuste decisioni per conseguire la vittoria al termine del campionato (quello inglese, data la provenienza del programma!).

Ovviamente il gioco comincia all'inizio di una nuova stagione, con la squadra situata nella quarta classifica (la più bassa).

Il programma presenta un menu dal quale è possibile selezionare varie opzioni (per esaminare il posto in classifica, la composizione della squadra e le caratteristiche dei calciatori).

I calciatori sono suddivisi in tre categorie: difensori, attaccanti e mezzala. Ovviamente, un buon manager tenta sempre di equilibrare le forze in campo.

Prima di ogni partita viene offerta la possibilità di riconfigurare la formazione in campo, sebbene nelle prime partite si abbiano appena 12 calciatori disponibili.

Nel corso del gioco, invece, il numero di calciatori può aumentare se si è in grado di acquistarne di nuovi sul mercato.

Ogni calciatore possiede tre caratteristiche principali, da tener ben presenti durante le contrattazioni. La prima è il livello di bravura (espresso da un valore da 1 a 5), la seconda è il "valore" di mercato che, pur essendo proporzionale al livello di bravura (5000 per ogni punto) può variare nel corso delle contrattazioni e la terza, infine, è l'"energia" del calciatore (da 1 a 20), che diminuisce dopo ogni partita.

Da tutto questo si nota che non bisogna schierare sempre i giocatori migliori, ma conviene eseguire delle rotazioni, affinché quelli che hanno appena giocato possano riposarsi e riacquistare le proprie forze.

All'inizio, quando i giocatori sono soltanto 12, questa economia di gioco è ovviamente impossibile.

Le contrattazioni avvengono sotto forma di asta, nella quale il giocatore tenta di accaparrarsi i calciatori migliori. Va però fatta attenzione al meccanismo di scambio: non è detto che offrendo una cifra consistente l'acquisto sia assicurato.

Talvolta, infatti, l'offerta viene rifiutata aumentando così il prezzo di mercato del calciatore.

Completate le contrattazioni ed eseguite le necessarie operazioni finanziarie ad esse connesse (ovviamente sempre fittizie), si passa a giocare una partita, dopo aver composto la formazione ritenuta più adeguata alle forze avversarie.

Il giocatore non interviene direttamente nella partita vera e propria (non è questo lo scopo del gioco): questa parte viene espletata dal computer, che visualizza di volta in volta brevi commenti sull'incontro.

Benché lo svolgimento grafico della partita non sia realizzato con le tecniche migliori (sul Micro BBC i calciatori sono tracciati in modo molto approssimativo), non per questo è meno appassionante.

Al termine della partita viene visualizzato il punteggio, assieme ai risultati delle altre squadre.

La posizione in classifica della squadra appartenente al giocatore viene aggiornata in base a tali risultati.

Il giocatore, quindi, deve cosiderare non soltanto la migliore strategia per ciascun incontro, ma deve anche saper programmare le migliori formazioni da adottare per le partite future, in particolare quelle con le squadre che conducono la classifica.

Ciò significa che occorre impiegare oculatamente i propri calciatori migliori, magari rischiando di perdere qualche punto all'inizio, per fronteggiare adeguatamente le formazioni più agguerrite.

Nonostante il gioco non possieda caratteristiche tipicamente "arcade", il suo successo è facilmente spiegabile: il giocatore prova con un certo realismo le emozioni di chi è alla guida di una squadra e, se poi riesce a portarla in serie "A", sente di aver raggiunto un obiettivo veramente difficile



LIZ HEANEY

L'attesa in panchina

Lo schermo qui riprodotto, tratto dalla versione di Football Manager per Commodore 64, mostra parte dell'azione di una partita. Durante questa fase il giocatore non ha possibilità d'intervento diretto: i risultati, infatti, dipendono dalle scelte fatte in precedenza e attendere la conferma delle proprie scelte è forse la parte più snervante del gioco

Football Manager: Per ZX81 e Spectrum Sinclair, Commodore 64 e Vic-20, Micro BBC, Electron, Amstrad e Dragon

Editori: Addictive Games Ltd. (GB)

Autore: Kevin Toms

Formato: Cassetta

Joystick: Non richiesti

Questioni di linguaggio

Nell'Intelligenza Artificiale molti dei risultati dipendono dal linguaggio usato

La maggior parte dei ricercatori interessati all'Intelligenza Artificiale impiegano il LISP o il PROLOG che, proprio per questo motivo, sono stati soprannominati "linguaggi AI". Tuttavia tale definizione può trarre in inganno poiché da un lato induce a pensare che il LISP e il PROLOG siano linguaggi non adatti a elaborazioni di tipo convenzionale e dall'altro che solo il LISP e il PROLOG, e non altri linguaggi di elaborazione simbolica come il POP-11 e lo SNOBOL4, consentano di scrivere efficaci programmi per l'Intelligenza Artificiale.
Il LISP è un "veterano" dei linguaggi di programmazione: sviluppato verso la fine degli anni '50 da John McCarthy al MIT, è pertanto "anziano" quanto il COBOL (vedere a pag. 1864).
Nel 1961 il linguaggio ha ricevuto il formato definitivo che conserva ancora oggi.
Il LISP è un linguaggio basato sull'elaborazione di "liste" e una delle strutture che riveste maggior importanza nell'ambito dell'Intelligenza Artificiale è la cosiddetta "lista di proprietà".
Ogni elemento di una lista, detto "atomo", possiede a sua volta una lista di proprietà, composta da una serie di associazioni "valore-attributo", che ne descrivono le caratteristiche e rappresentano una specie di database per quell'elemento. Le liste di proprietà vengono manipolate con le funzioni GET e PUTPROP. Ad esempio, le istruzioni:

```
(PUTPROP 'IERI 0.1 'PRECIPITAZIONI)
(PUTPROP 'DOMANI 'COPERTO 'PREVISIONE)
```

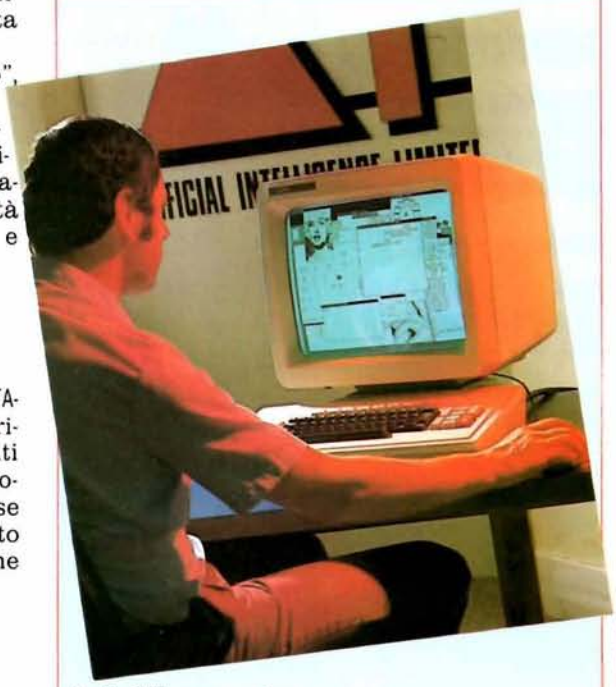
assegnano il valore 0.1 all'attributo PRECIPITAZIONI dell'atomo IERI e il valore COPERTO all'attributo PREVISIONE dell'atomo DOMANI. Gli attributi PRECIPITAZIONI e PREVISIONE sono quindi paragonabili a campi di un record in un database (vedere a pag. 1577). Il valore di ogni attributo può essere individuato usando la funzione GET.
Ad esempio:

```
(SETQ PIOGGIA (GET 'IERI 'PRECIPITAZIONI))
```

estrae il valore PRECIPITAZIONI dalla lista di proprietà IERI e lo assegna come nuovo valore-attributo all'atomo PIOGGIA, anche se, in realtà, il valore di un atomo è un particolare genere di proprietà utilizzabile senza dover ricorrere a GET e PUTPROP.
Le liste di proprietà consentono di realizzare complesse ma elastiche strutture dati, spesso adoperate per imitare il funzionamento della memoria umana.

Programmazione AI in PROLOG

Il PROLOG è la principale alternativa al LISP per la programmazione AI.
Essendo un linguaggio "dichiarativo", consente al programmatore di specificare fatti e regole da poter impiegare per rispondere a domande riguardanti specifici argomenti e relazioni.
Sinteticamente, il PROLOG risponde ai quesiti ricorrendo ad un metodo di dimostrazione basato sul "principio di risoluzione", in cui la proposizione che si vuol dimostrare viene in effetti negata. Questa è una versione sofisticata della tecnica di "dimostrazione per assurdo", applicata a un database.
Come il LISP, il PROLOG è un linguaggio



Ambienti di programmazione
La rete Interlisp-D, progettata dalla Rank Xerox, offre ai ricercatori AI un ambiente di programmazione completo, comprendente una grafica interattiva, svariate procedure di debugging e una serie di funzioni d'archiviazione "in linea" sfruttando un'unità a dischi da 28 Megabyte. Abbinando l'Interlisp-D al sistema di "programmazione delle conoscenza" LOOPS, il programmatore può realizzare economici sistemi di conoscenza, applicando una combinazione di diversi stili di programmazione

Per concessione della "ARTIFICIAL INTELLIGENCE LTD"

molto flessibile, che possiede funzioni per la creazione di complicate strutture di dati (comprese le liste) e per operazioni di ingresso/uscita, calcolo, gestione dei file ecc.
Oltre al LISP e al PROLOG, esistono altri linguaggi utilizzati nell'Intelligenza Artificiale: tra questi il POP-11 (impiegato soprattutto all'interno dell'ambiente POPLOG ideato alla Sussex University) o altri di uso più comune come il C, il PASCAL e, addirittura, anche il BASIC. Tuttavia questi linguaggi implicano un maggior impegno di programmazione.
Oltre alla scelta del linguaggio, anche altri fattori contribuiscono alla realizzazione di buoni programmi per l'Intelligenza Artificiale: tra questi l'impiego di "ambienti AI" come il LOOPS e il POPLOG, gestiti da apposite "stazioni di lavoro" per Intelligenza Artificiale.
Il LOOPS (LISP Object-Oriented Programming System), sviluppato nel Centro di ricerca della Xerox a Palo Alto e basato su un dialetto del LISP chiamato INTERLISP-D, offre all'utente una vasta scelta di strumenti di programmazione. Alcuni di questi sono di tipo convenzionale (come la gestione delle "finestre" e delle "icone"), altri sono invece più sofisticati (come le procedure d'inferenza pre-programmate da applicare ai sistemi esperti). L'intero sistema viene pilotato da una speciale stazione di lavoro AI che contiene un processore predisposto per elaborare il LISP.
Il POPLOG gira sui computer della serie VAX quindi non richiede un apposito hardware ma, dal punto di vista del software, somiglia molto al LISP.
Offre una serie di utili pacchetti applicativi, tra cui un programma di "editor" per lo schermo e consente di chiamare il LISP o il PROLOG dall'interno del POP-11. Inoltre incorpora una biblioteca per il confronto di configurazioni e altre procedure appositamente modificate per le applicazioni dell'Intelligenza Artificiale.
Il principale difetto di questi sistemi è l'impossibilità di trasferirne su altri apparecchi (in particolare gli home computer) le applicazioni, dal momento che vengono sviluppate all'interno di uno specifico ambiente operativo.
La realizzazione di applicazioni per l'Intelligenza Artificiale pone problemi molto complessi che spesso sono risolti solo durante la stessa fase di programmazione. Quindi l'obiettivo di uno studioso di AI è quello di sviluppare rapidamente un "prototipo" di base, da perfezionare gradualmente introducendo piccole modifiche.
Le ricerche sull'Intelligenza Artificiale hanno dimostrato che i comuni strumenti offerti dal software e le tradizionali metodologie di progettazione non consentono di affrontare in modo adeguato i problemi proposti dalla programmazione AI: il settore è quindi aperto a nuove tecniche e a nuove idee.

LINGUAGGI A CONFRONTO

Nel corso degli anni sono stati ideati numerosi linguaggi che consentono al programmatore AI di sviluppare progetti sempre più ambiziosi. Tali linguaggi possiedono, anche se in misura diversa, alcune caratteristiche comuni:

1) La possibilità di programmare procedure ricursive.

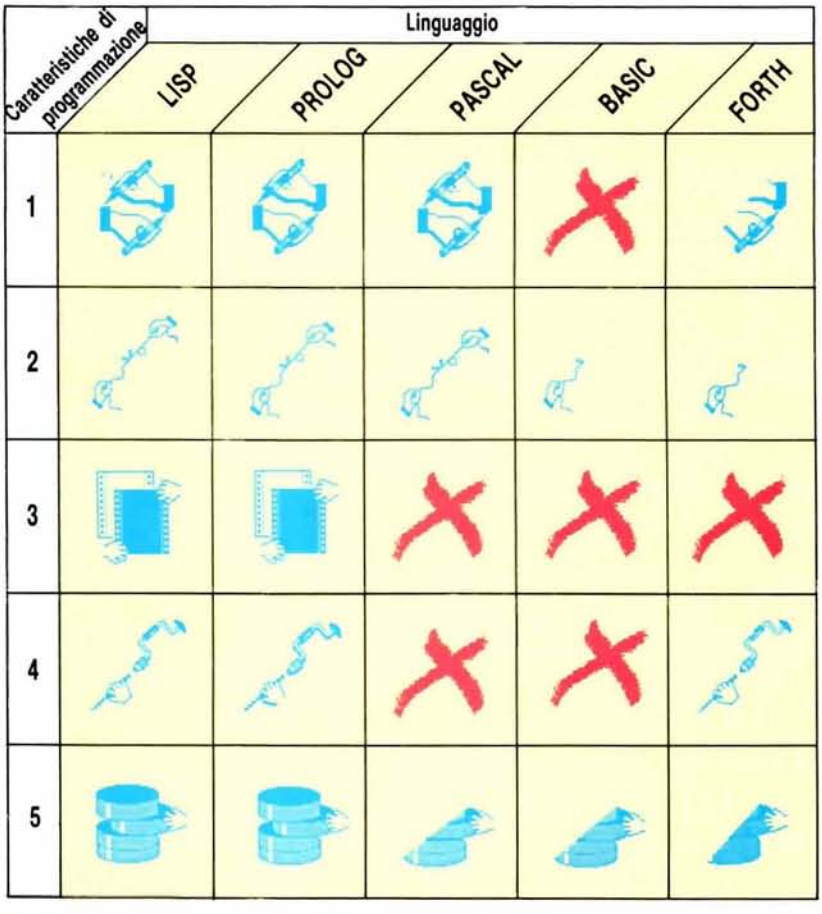
2) La presenza di efficienti funzioni per la gestione delle stringhe e la manipolazione dei simboli, che permettano di creare strutture di dati flessibili e di varia complessità.

3) Una discreta uniformità tra dati e programma (il LISP e il PROLOG usano un identico formato sia per i programmi sia per i dati).

4) La facoltà di ampliare la sintassi (il LISP e il PROLOG, teoricamente, permettono di creare un nuovo linguaggio partendo da quello originale).

5) La possibilità di accedere a un database interno (il PROLOG dispone di un proprio database relazionale, mentre il LISP ricorre alle liste di proprietà).

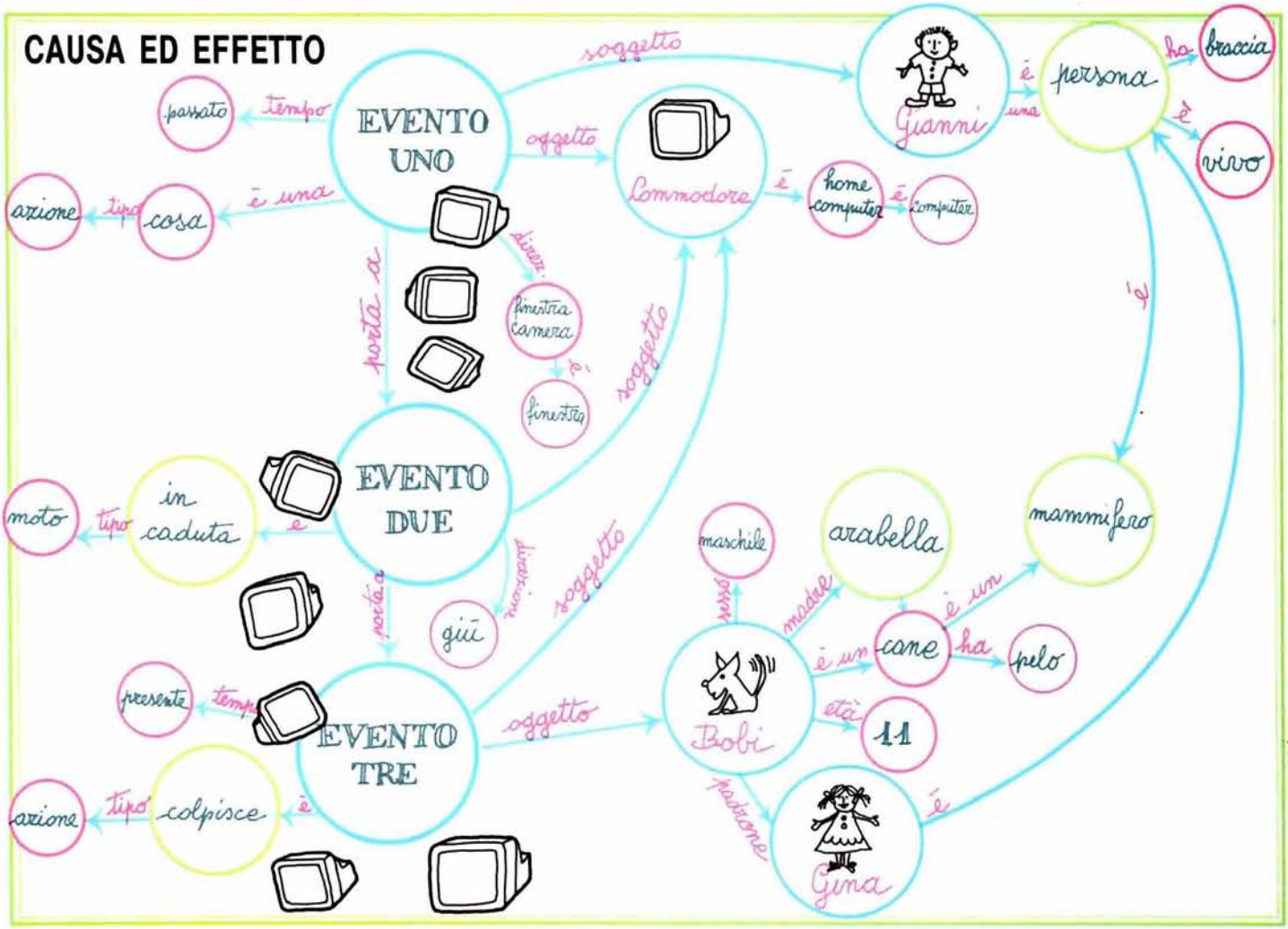
Come in altri settori dell'elaborazione, anche nell'Intelligenza Artificiale occorrono tutti i vari strumenti che facilitano la programmazione (programmi di "editor", routine grafiche, procedure di documentazione o per la verifica della sintassi e degli errori, compilatori, ecc.).
La tabella comparativa qui a fianco mostra la distribuzione delle precedenti caratteristiche in 5 linguaggi di alto livello



IAN McKINNELL

Rappresentare la conoscenza

Il metodo adottato per schematizzare le informazioni è di cruciale importanza in AI



Gli esperti in Intelligenza Artificiale tentano di imitare sistemi complessi quali la memoria o i meccanismi di ragionamento dell'uomo. Per questo motivo in AI si parla di "rappresentazione della conoscenza" e non di "rappresentazione di dati".

In genere i programmatori hanno un'idea piuttosto precisa del significato della parola "dato" ma sono scettici sull'uso del termine "conoscenza" per indicare le informazioni immagazzinate in un computer. In realtà la differenza tra le due definizioni risiede nella natura dei problemi affrontati e delle soluzioni proposte. Vediamo quali sono i quattro elementi principali che permettono di distinguere la "conoscenza" dal semplice "dato".

Innanzi tutto, le strutture usate per rappresentare la conoscenza devono essere flessibili, ossia capaci di variare le proprie dimensioni (tramite un'assegnazione dinamica della memoria) durante l'esecuzione del programma. Sfortunatamente molti dialetti BASIC non prevedono l'impiego di simili strutture dinamiche.

In secondo luogo, la rappresentazione della conoscenza richiede strutture a più livelli o "multi-strato" e non a livello singolo come nel caso dei dati. Le liste, ad esempio, devono poter contenere delle sottoliste, gli alberi dei sottoalberi, le regole potersi riferire a delle sottoregole e così via.

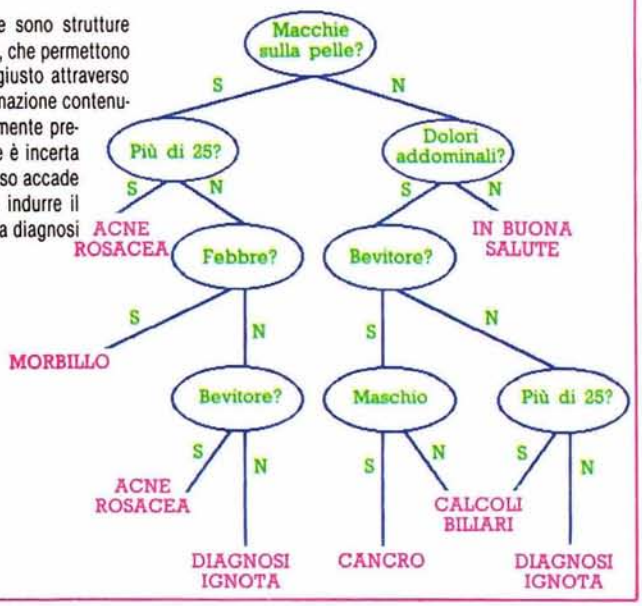
Inoltre le rappresentazioni della conoscenza di solito contengono tipi di dati eterogenei. In un programma di gioco, ad esempio, il record che descrive un personaggio deve poter accogliere stringhe (ad es. un nome), valori numerici (ad es. età e statura), puntatori ad altri record (ad es. quelli relativi ad alleati e avversari del personaggio), regole che ne descrivono il comportamento in particolari

Sequenza semantica

Le reti semantiche vengono usate per collegare azioni e oggetti e per descriverne le correlazioni. La rete qui riprodotta significa: "Gianni getta dalla finestra il suo computer e colpisce il cane di Gina". I cerchi e le frecce rappresentano rispettivamente i nodi della rete e gli archi di connessione. Dai collegamenti semantici si può dedurre, ad esempio, che «Il computer colpisce un mammifero con pelliccia» oppure che «Qualcuno colpisce il cucciolo di Arabella». In un vero sistema, la rete è di solito molto più estesa di quella rappresentata in figura

SISTEMA DIAGNOSTICO

Gli alberi di decisione sono strutture semplici ma funzionali, che permettono di seguire il sentiero giusto attraverso l'albero solo se l'informazione contenuta nei nodi è estremamente precisa. Se l'informazione è incerta o ambigua (come spesso accade nella realtà) essa può indurre il sistema a emettere una diagnosi totalmente sbagliata



CAROLINE CLAYTON

situazioni e via dicendo. La realizzazione di simili strutture è impossibile in BASIC, o perlomeno molto complessa, poiché le matrici possono contenere o solo stringhe o solo numeri, per non parlare poi di matrici, subroutine o elementi d'altro tipo.
Infine, mentre le rappresentazioni della conoscenza sono di tipo "attivo", le strutture dei dati sono "passive". Quindi, nei sistemi basati sulla conoscenza, alla suddivisione in "programma" e "dati" è stata sostituita una struttura a tre livelli composta da: fatti, regole e motore d'inferenza. Mentre i fatti corrispondono ai dati e il motore d'inferenza al programma, le regole rappresentano una via di mezzo tra queste strutture e vengono spesso chiamate anche "dati attivi".
Dopo aver ricordato le principali differenze tra i dati e la conoscenza, si possono esaminare alcune strutture usate per rappresentare e memorizzare la conoscenza.

Le matrici

Le matrici permettono di rappresentare la conoscenza in modo semplice, ma la loro struttura è troppo elementare per le applicazioni dell'Intelligenza Artificiale. Si immagini, ad esempio, un sistema esperto semplificato per diagnosi mediche, articolato solo su 7 sintomi (fatti) e 4 diagnosi (ipotesi). In teoria dovrebbe essere possibile codificarne la conoscenza all'interno di una matrice bidimensionale. Questa dovrebbe consentire l'applicazione di una semplice strategia d'inferenza che raccoglie i dati (fatti), un elemento alla volta, li inserisce nella matrice usando una scala di rappresentazione del tipo: -1 (no), 0 (possibile), +1 (sì) e moltiplica i valori contenuti in ciascuna colonna. La colonna che produce il totale più elevato rappresenta l'ipotesi più probabile.

In pratica però tale struttura si dimostra poco flessibile. Infatti, se il sistema diagnostico viene ampliato a 200 sintomi e 100 tipi di malattie, la matrice corrispondente dovrebbe contenere ben 20.000 celle, molte delle quali rimarrebbero vuote. Inoltre il formato eccessivamente "appiattito" di una matrice non è adatto alla struttura gerarchica delle conoscenze mediche che, basandosi sui sintomi, permettono di escludere interi gruppi di malattie. Quindi la struttura più adeguata per un sistema esperto di questo tipo è un "albero di conoscenza".

Gli alberi

Le strutture ad albero sono di grande aiuto nella programmazione dell'Intelligenza Artificiale. In questo ambito, due importanti applicazioni sono: 1) gli Alberi di Decisione e 2) le Gerarchie di Discendenza. Vediamo, ad esempio, come sia possibile riorganizzare la conoscenza del sistema diagnostico, visto nell'esempio precedente, all'interno di un albero di decisione (vedere figura).
Questa struttura produce risultati affidabili purché le verifiche eseguite sui nodi forniscano risposte precise. Se invece le risposte hanno un certo grado di ambiguità, il sistema può facilmente imboccare una diramazione sbagliata.
Gli alberi a gerarchia di discendenza collegano i fatti a "concetti" e ne riproducono le correlazioni in modo ordinato, consentendo di trarre deduzioni logiche.

Le reti semantiche

Una rete semantica è più complessa di un albero. Mentre gli alberi possiedono uno speciale nodo che funge da "radice" e tutti i suoi rami si aprono a ventaglio in una sola direzione (di solito verso il basso), le reti semantiche presentano più nodi "radice" e linee di collegamento o "archi" che puntano in tutte le direzioni.
Queste strutture vengono molto usate nell'Intelligenza Artificiale per rappresentare la conoscenza relativa al linguaggio e concernente azioni ed eventi. In una rete semantica, i nodi e gli archi rappresentano rispettivamente gli oggetti e le relazioni che li collegano.
Simili reti sono una generalizzazione delle liste di proprietà del LISP (vedere a pag. 1864). Poiché entrambe codificano l'informazione in forma di coppie "valore-attributo" riferite a un oggetto, possono essere classificate sotto un'unica categoria, detta "tupla". Una "n-tupla" indica semplicemente un gruppo di "n" oggetti. Le strutture "oggetto-valore-attributo" (come le reti semantiche e le liste di proprietà) vengono quindi rappresentate con triplette, cioè "3-tuple", del tipo:

Oggetto	Attributo	valore
BOBI	E' - UN	CANE

La rappresentazione della conoscenza per mezzo di triplette si avvicina molto a quella delle cosiddette "cornici" (in inglese "frame"), usatissime nella programmazione AI.
Una "cornice" è una struttura che contiene un numero variabile di "slot", concettualGFmente simili ai campi di un record convenzionale.
In pratica, le cornici sono strutture "n-tuple" dove "n" può variare.
La seguente rappresentazione a cornici descrive alcune informazioni della rete semantica presentata più avanti:

Nome:	Evento-3
Tipo:	Colpisce
Tempo:	Presente
Soggetto:	Commodore
Oggetto:	Bobi
Antecedente:	Evento-2

Nome:	Bobi
E' un:	—>Prototipo-Cane
Padrone:	Gina
Tipo:	Essere animato
Sesso:	Maschile
Età:	11
Genitori:	(Ferdinando, Arabella)
Coinvolto in:	(Evento-3, Evento-7, Evento-20...

Nome:	Prototipo-Cane
Ha:	Pelliccia
Può:	(Abbaiare, Mordere, Cacciare...)
Zampe:	Valore per difetto = 4
Esempi:	(Bobi, Buck, Arabella, Pippo)
Pericoloso:	Se (IF) è piccolo e (AND) addormentato o (OR) dimena la coda allora (THEN) No altrimenti (ELSE) Sì
Azione:	Se (IF) addormentato allora (THEN) non disturbarlo

Questi esempi mostrano che le "slot" possono accogliere vari tipi di dato: Età contiene un numero, Genitori una lista, Pericoloso una regola di decisione, E' un contiene un puntatore a un'altra cornice e così via.
Le "slot" corrispondono agli attributi di una lista di proprietà o agli archi di una rete semantica.

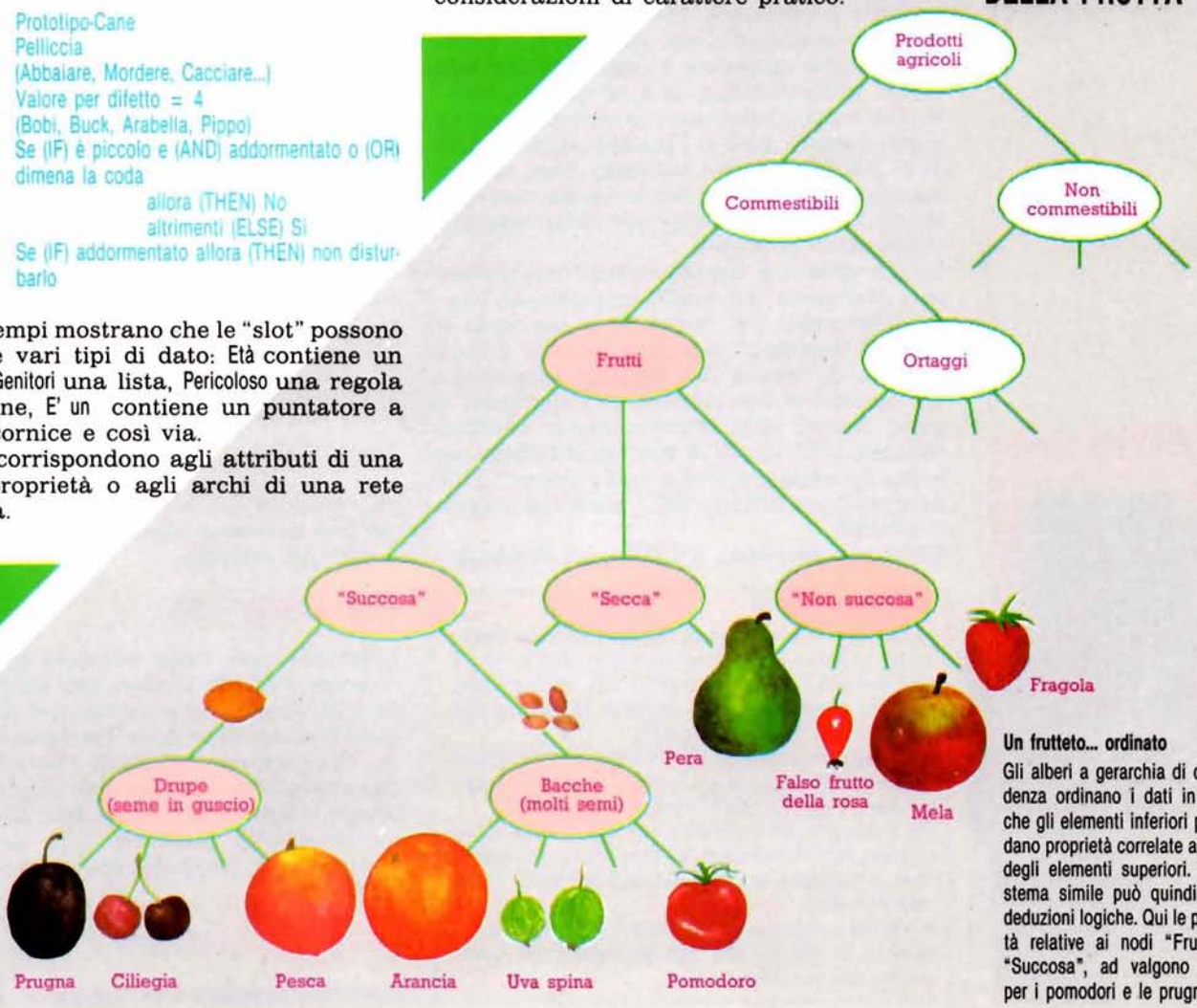
La logica della rappresentazione

Poiché le informazioni possono essere rappresentate con strutture di diverso tipo, purché sufficientemente flessibili, la scelta cade su quella più efficiente. Tuttavia, alcuni studiosi dell'Intelligenza Artificiale preferiscono affidarsi a una tecnica di rappresentazione più generale, chiamata "logica predicata". La logica, intesa come linguaggio di rappresentazione, costituisce il nucleo centrale del PROLOG che poggia su strutture di programmazione dette "clausole a corna". Per esempio, la procedura:

```
pericoloso(X):—
                cane(X), non (innocuo(X))
innocuo(X):—
                piccolo(X), addormentato(X)
innocuo(X):—
                dimena la coda(X)
```

è una versione PROLOG di una delle regole riguardanti i cani rappresentate nelle precedenti strutture a cornice.
Quindi, sebbene in teoria la logica rivesta un'importanza maggiore rispetto alle strutture finora descritte, in realtà la scelta della rappresentazione della conoscenza nel campo dell'Intelligenza Artificiale dipende da considerazioni di carattere pratico.

CLASSIFICAZIONE DELLA FRUTTA



DAVID DAVIES

Un frutteto... ordinato
Gli alberi a gerarchia di discendenza ordinano i dati in modo che gli elementi inferiori possiedano proprietà correlate a quelle degli elementi superiori. Un sistema simile può quindi trarre deduzioni logiche. Qui le proprietà relative ai nodi "Frutta" o "Succosa", ad valgono anche per i pomodori e le prugne

Il mondo delle liste

Inizia una serie dedicata alle caratteristiche del linguaggio LISP

La crescente popolarità del LISP è dovuta soprattutto al suo impiego nelle ricerche e nello sviluppo dell'Intelligenza Artificiale. Via via che l'interesse per questo linguaggio aumenta, il LISP assume sempre più il ruolo di "linguaggio universale" per una vasta serie di applicazioni.
Tale situazione ha favorito la comparsa di numerose versioni, dedicate alle più diverse applicazioni: dai sistemi operativi alla scrittura di compilatori e di programmi di gioco. Logica conseguenza di questa proliferazione è stata la comparsa di molti dialetti LISP. Tuttavia, grazie alla struttura semplice e lineare del LISP, la maggior parte delle versioni conserva una discreta standardizzazione che facilita il trasferimento del linguaggio da un apparecchio all'altro.
Uno dei maggiori problemi che si incontrano programmando in LISP è l'assenza di alcune funzioni o istruzioni, dovuta alla mancanza di uno standard ufficiale. Tuttavia è estremamente facile estendere il linguaggio introducendo, all'occorrenza, le nuove istruzioni.
Il LISP è completamente diverso dai linguaggi più comuni quali il PASCAL, il FORTRAN e il BASIC, poiché possiede una sintassi semplice e uniforme che lo rende uno strumento ideale per manipolare informazioni e confrontare problemi.
La sua struttura, basata su liste interconnesse, è facilmente adattabile ai principali sistemi informativi del computer e consente di usare il linguaggio per operazioni di ordinamento e di ricerca, per funzioni aritmetiche, ma anche per l'esecuzione di programmi di gioco. Inoltre molti microcomputer adottano package LISP dotati di speciali istruzioni per le sue funzioni grafiche e audio (come il LISP Acornsoft per il Micro BBC, usato nei seguenti esempi).
Come già ricordato, il LISP è un linguaggio basato su liste, per molti aspetti simili alle matrici ma che, diversamente da queste, non hanno una lunghezza fissa.
Le liste devono sempre essere racchiuse tra parentesi tonde, seguendo il formato:

```
(a b c d e ...)
```

dove a, b, c, ecc. sono gli elementi o "atomi" della lista. Questi atomi possono contenere dati numerici, non numerici (caratteri o variabili) oppure altre liste. Si noti che, nella rappresentazione di una lista, gli elementi sono separati da spazi e non da virgole.
Le liste vengono elaborate per mezzo di funzioni in modo molto simile a quello usato in BASIC con DEF FN, che esegue un'operazione sui suoi argomenti per produrre un risultato. Le funzioni hanno il seguente formato:

```
(funz a b c d ...)
```

dove funz rappresenta il nome della funzione e a, b, c... sono i suoi argomenti. Come mostra l'esempio, la funzione sembra essere il primo elemento della lista. Si consideri una lista che contiene i primi sei numeri "primi":

```
(1 2 3 5 7 11)
```

Volendo sommare questi numeri in BASIC, va usata la seguente espressione:

```
1 + 2 + 3 + 5 + 7 + 11
```

Il LISP, invece, ricorre alla funzione plus:

```
(plus 1 2 3 5 7 11)
```

che produce come risultato 29. La funzione plus può elaborare qualsiasi numero di argomenti. Ad esempio:

```
(PLUS 1 2 3 (PLUS 5 7 11))
```

In questo caso, viene calcolata per prima la funzione PLUS più interna, che dà come risultato 23. Questo valore viene poi usato come quarto argomento della funzione PLUS esterna, che produce il risultato finale 29. L'esempio precedente contiene due funzioni concatenate, eseguite in successione. L'introduzione di un'altra funzione, SETQ, permette di assegnare il risultato dell'espressione alla variabile A:

```
(SETQ A (PLUS 1 2 3 5 7 11))
```

dove SETQ possiede due argomenti: la variabi-



Implementazioni di LISP

Nonostante il crescente successo di questo linguaggio, il mercato offre un numero limitato di pacchetti LISP adatti ai micro. Il LISP Acornsoft è disponibile per il Micro BBC e l'Electron della Acorn ma, in entrambi i casi, è necessario acquistare a parte anche il relativo manuale per l'utente.
Chi possiede microcomputer dotati funzionanti sotto CP/M ha una scelta un poco più ampia: il Toolworks Lisp/80, l'iLISP, il più economico muLISP-83 della Microsoft e altri ancora.
Questi interpreti LISP funzionano sulla maggior parte di computer basati sul CP/M (Memotech, Einstein, ecc.) ma non sull'Amstrad, che possiede una memoria troppo limitata (il minimo richiesto è 48 K).
Il pacchetto migliore è probabilmente il Toolworks che può archiviare nei suoi 48 Kbyte circa 3600 celle di lista e 11000 caratteri per i nomi degli atomi

le A e una funzione che produce come risultato l'intero 29. Naturalmente anche la funzione SETQ fornisce un risultato (in questo caso il valore 29). Quindi l'assegnazione seguente, non consentita da molti dialetti BASIC:

```
LET B = 1 + 2 + 3 + 5 + 7 + 11
```

può essere programmata in LISP come:

```
(SETQ B (SETQ A (PLUS 1 2 3 5 7 11)))
```

Se si inserisce la nuova funzione TIMES è possibile assegnare 2 volte il valore di A alla variabile B:

```
(SETQ B (TIMES 2 (SETQ A (PLUS 1 2 3 5 7 11))))
```

Anche in questo caso viene valutata per prima la funzione PLUS il cui risultato (29) viene assegnato alla variabile A dalla funzione SETQ più interna.
Questa, a sua volta, produce il valore 29 che rappresenta il secondo argomento di TIMES. La funzione TIMES calcola il nuovo risultato 58 e lo passa alla SETQ più esterna, che lo assegna infine alla variabile B.
L'intera espressione, dunque, dà come risultato 58 che può essere usato come argomento in ulteriori funzioni.
Si noti che, nell'ultimo esempio, il numero di parentesi tonde è aumentato notevolmente. Questa è una fastidiosa caratteristica del LISP ma, se il programma è ben progettato, è abbastanza facile abbinare correttamente il numero delle parentesi.
Inoltre alcuni sistemi LISP facilitano il lavoro indicando il numero delle parentesi che devono essere chiuse: il LISP Acornsoft, per il Micro BBC visualizza all'inizio della linea un numero di frecce pari a quello delle parentesi tonde ancora aperte. La funzione TIMES introdotta nell'ultimo esempio possiede due argomenti: l'intero 2 e un'espressione che dà come risultato 29. Poiché il numero degli argomenti di TIMES e di PLUS può variare, il LISP considera valide le seguenti espressioni:

```
(TIMES 1 2 4 8 16)
(TIMES 1 2 4 8 (TIMES 4 4))
(TIMES 1 2 (TIMES 2 2)(TIMES 2 4)(PLUS 8 8))
```

che danno tutte come risultato 1024.
Tuttavia alcune versioni del linguaggio limitano il numero di argomenti consentiti per questi tipi di funzione. Il LISP Acornsoft, ad esempio, accetta un massimo di 28 argomenti, ma altre versioni pongono limiti ancora più restrittivi. Gli esempi finora esaminati dimostrano che un'istruzione LISP è semplicemente una lista il cui primo elemento è la funzione da elaborare e quelli successivi rappresentano gli argomenti della stessa funzione. Questi possono essere a loro volta liste il cui primo argomento è una funzione che genera un risultato. Vediamo adesso come procedere per creare una lista da usare, ad esempio, come un titolo. Ovviamente non è possibile scrivere semplicemente:

Liste contro Matrici

Le liste offrono due importanti vantaggi sulle matrici. Innanzi tutto non richiedono che venga riservata loro una quantità "a priori" di memoria e inoltre sono strutture "dinamiche", ossia di lunghezza variabile, che possono contrarsi o espandersi per accogliere i dati durante la stessa esecuzione del programma. Infine le liste si adattano meglio delle matrici alle procedure ricursive, comunemente usate nella programmazione dell'Intelligenza Artificiale




```
'(CORSO DI PROGRAMMAZIONE LISP)
```

come una normale lista a quattro elementi: in tal caso il LISP tenterebbe di valutare il primo elemento CORSO come un nome di funzione seguito da tre argomenti (DI, PROGRAMMAZIONE e LISP), considerando ciascun argomento come un nome di variabile.
Per comunicare al LISP di non valutare l'espressione, è necessario far precedere la lista da una virgoletta (').

```
(CORSO DI PROGRAMMAZIONE LISP)
```

La prossima istruzione assegna la precedente lista di quattro elementi non numerici alla variabile MAG.

```
(SETQ MAG '(CORSO DI PROGRAMMAZIONE LISP))
```

Si noti che il LISP non prevede variabili di diverso tipo: mentre il BASIC impone il formato A% per le variabili intere, A per quelle reali e A$ per le stringhe, il LISP non fa distinzione tra le variabili. Le seguenti espressioni, pertanto, sono tutte valide:

(SETQ A 3)	A = l'intero 3
(SETQ A 'LISP)	A = la stringa 'COMPUTER'
(SETQ A (PLUS 2 4 8))	A = il risultato di 2+4+8
(SETQ A '(PROGRAMMAZIONE LISP))	A = la lista (PROGRAMMAZIONE LISP)
(SETQ A B)	A = il valore della variabile B

Benché i tipi di variabile "intero" e "stringa" siano completamente standard, solo pochi micro possiedono una versione LISP per l'aritmetica in virgola mobile. Nella maggior parte dei casi l'aritmetica intera è sufficiente e, nell'improbabile eventualità che sia necessaria l'aritmetica in virgola mobile, la si può simulare usando gli interi standard.
In seguito vengono descritte le funzioni per la manipolazione degli argomenti e i principi di ricursione, fulcro della grande potenza d'elaborazione del LISP.

Linee di melodia

Il progetto dell'interfaccia MIDI prosegue: il collegamento del chip ACIA

Il "cuore" dell'interfaccia MIDI è il chip ACIA (Asynchronous Communications Interface Adaptor) compatibile con i processori 6502 e 6510 usati, rispettivamente, nel Micro BBC e nel Commodore 64. È quindi possibile collegare le linee di controllo, d'indirizzamento e dei dati del processore direttamente ai piedini di questo chip.
La scheda per il Commodore 64 è progettata per l'innesto alla porta d'espansione del computer, mentre quella per il Micro BBC viene collegata alla porta "tube", situata sul retro della macchina, tramite un cavo a nastro da 40 canali dotato di un apposito connettore. I collegamenti necessari per le due versioni del progetto sono descritti separatamente. Realizzata la scheda, occorre verificarne il funzionamento eseguendo alcune prove quali il caricamento di dati nel chip ACIA e l'esecuzione di un semplice ciclo di controllo.
Per accedere alle periferiche collegate al bus dei dati principale, ma che non prevedono lo stesso numero di linee d'indirizzo rispetto alla CPU, è necessaria una decodifica degli indirizzi. Di solito tali periferiche possiedono un piedino di "selezione del chip" che, se selezionato correttamente, consente di usare il bus dei dati; di conseguenza le linee di indirizzamento collegate alla periferica permettono la selezione dei diversi registri.
I segnali di selezione del chip, per gli svariati apparecchi collegati al sistema, vengono generati decodificando le linee più significative degli indirizzi: le combinazioni di bit inviate su di esse possono essere quindi impiegate per selezionare una particolare periferica. In questo modo i registri interni dell'apparecchio entrano a far parte della mappa di memoria della CPU ed è possibile accedervi e manipolarli come se fossero normali locazioni di memoria.
Ovviamente ci si deve accertare che gli indirizzi usati dalla periferica non corrispondano ai registri adoperati dal sistema operativo del computer.
La porta d'espansione del Commodore 64 possiede 2 uscite, indicate dalle sigle I/01 e I/02. Queste linee vengono poste a livello logico "basso" quando il sistema accede alle pagine $DE e $DF. Collegando la linea CS2 del chip ACIA all'uscita I/01 è possibile "impaginare" il chip nella pagina di memoria $DE. Poiché l'ACIA non è collegato alle linee d'indirizzo A1-A7, i suoi registri interni possono essere raggiunti usando un qualsiasi indirizzo compreso tra $DE00 e $DEFF. Collegando direttamente A0 al piedino di selezione del registro dell'ACIA, tutti gli indirizzi di valore pari permettono di accedere ai registri di trasmissione/ricezione dei dati. La scelta più ovvia, comunque, è quella degli indirizzi $DE00 (56832 decimale) e $DE01 (56833 decimale).
Il Modello B del Micro BBC offre due possibili porte di collegamento per l'interfaccia, ognuna con propri pregi e difetti. Da un lato la porta "tube" permette di accedere direttamente al bus dei dati a 2MHz e fornisce una linea di decodifica d'indirizzo (NTUBE) per gli indirizzi &FEE0 e &FEFF; dall'altro il Micro BBC ha il difetto di leggere particolari indirizzi del "tube", appena l'apparecchio viene acceso, per controllare se la porta "tube" è collegata a qualche periferica. Se però il sistema non è collegato al secondo processore, il computer si blocca perché attende di ricevere informazioni da tale circuito.
Questo inconveniente può essere aggirato connettendo una delle linee d'indirizzo di ordine alto al CS0 del chip ACIA oppure collegando la linea NTUBE a CS2 e innestando la scheda nella porta "tube", ma solo dopo aver acceso il computer: è questa la soluzione adottata nel nostro progetto. Una terza alternativa prevede l'impiego del bus a 1MHz. I piedini indicati dalle sigle NPGFC e NPGFD accedono alle periferiche attraverso le pagine di memoria &FC e &FD e vengono usati come le linee d'uscita I/01 e I/02 nella versione per il Commodore 64.
L'impiego del bus a 1MHz presenta però due svantaggi. In primo luogo, poiché il clock del sistema viene rallentato a 1MHz, sulle due linee di decodifica possono comparire segnali "spuri", il che costringe a impiegare un "circuito di ripulitura". In secondo luogo sul bus da 1 MHz manca una linea d'alimentazione a 5 volt, pertanto questa tensione deve essere prelevata dalla porta utente oppure dalla presa d'alimentazione ausiliaria.

Programmazione dell'ACIA

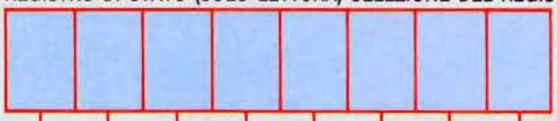
Per programmare l'interfaccia MIDI si deve conoscere il funzionamento dei quattro registri del chip ACIA. Quanto segue si riallaccia alla descrizione iniziata a pagina 1825 e si riferisce alle funzioni dei bit dei registri di controllo e di stato. Intervenendo sul registro di controllo si può programmare il chip ACIA in modo che esso generi un'interruzione per la CPU, qualora vengano selezionati particolari bit di stato all'interno dei circuiti di ricezione e/o trasmissione dell'ACIA. Quando i bit 5 e 6 del registro di controllo vengono posti rispettivamente a 1 e a 0 e il bit di stato TDR si trova a livello basso, il chip ACIA genera una interrupt di trasmissione. L'interrupt viene annullata con una scrittura di dati nel TDR. Se invece vengono posti a 1 il bit 7 del registro di controllo 7 e il bit 0 del registro di stato, l'ACIA genera una interrupt di ricezione. Questa viene annullata con una lettura del registro di ricezione dei dati (RDR), purché non si trovi a livello alto il bit 5 del registro di stato, che viene azzerato leggendo prima lo stesso registro di stato poi l'RDR. Anche

I REGISTRI DELL'ACIA

REGISTRI DI TRASMISSIONE/RICEZIONE DEI DATI:
SELEZIONE DEL REGISTRO = 1

RICEZIONE - LETTURA DEL REGISTRO
TRASMISSIONE - SCRITTURA DEL REGISTRO

REGISTRO DI STATO (SOLO LETTURA) SELEZIONE DEL REGISTRO = 0



- REGISTRO DI RICEZIONE DEI DATI PIENO
- REGISTRO DI TRASMISSIONE DEI DATI VUOTO
- INDIVIDUA LA PORTANTE DEI DATI
- PRONTO PER LA TRASMISSIONE
- ERRORE DI FORMATO
- SOVRACCARICO DEL RICEVITORE
- ERRORE DI PARITÀ (*)
- RICHIESTA D'INTERRUZIONE

REGISTRO DI CONTROLLO (SOLO SCRITTURA) SELEZIONE DEL REGISTRO = 0



SELEZIONE DEL CONTATORE DI DIVISIONE
00 DIVIDE PER 1 (*)
01 DIVIDE PER 16 (*)
10 DIVIDE PER 64
11 AZZERAMENTO PRINCIPALE

SELEZIONE DELLA PAROLA

	BIT DEI DATI	BIT DI STOP	PARITÀ
000	7	2	PARI (*)
001	7	2	DISPARI (*)
010	7	1	PARI (*)
011	7	1	DISPARI (*)
100	8	2	NESSUNA (*)
101	8	1	NESSUNA
110	8	1	PARI (*)
111	8	1	DISPARI (*)

CONTROLLO DELLE INTERRUPT DI TRASMISSIONE
00 DISABILITA (*)
01 ABILITA
10 DISABILITA (*)
11 DISABILITA (*)

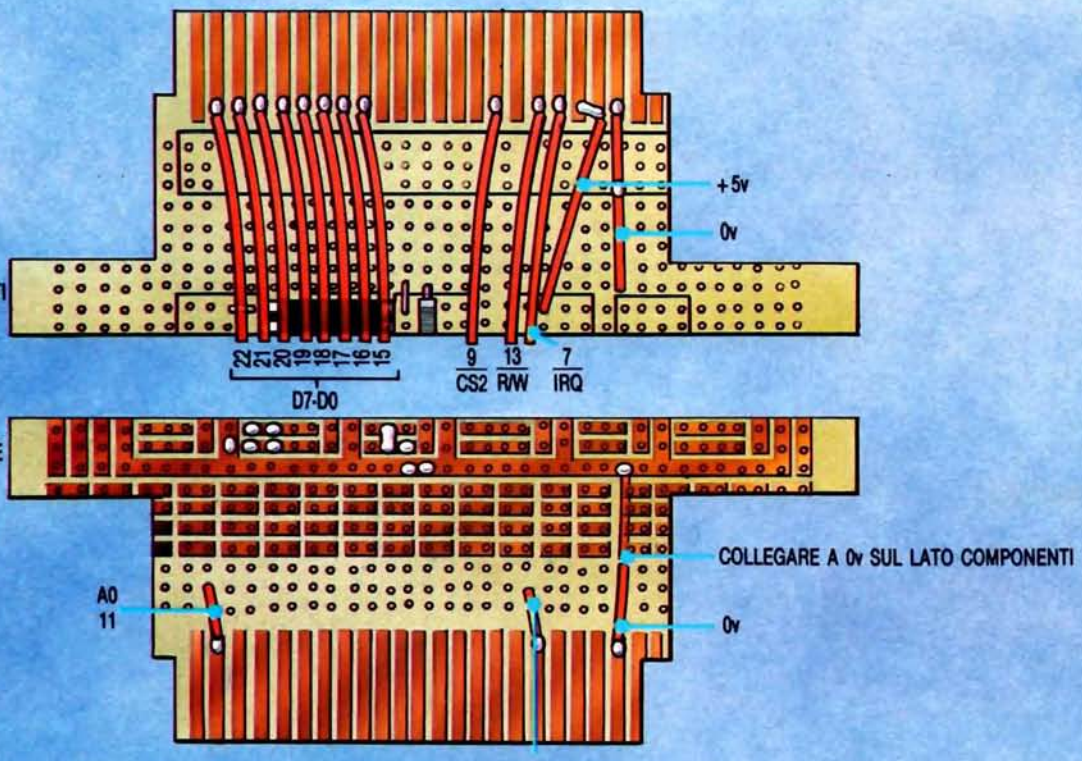
CONTROLLO DELLE INTERRUPT DI RICEZIONE
0 DISABILITA
1 ABILITA

(*) = NON APPLICABILE AL PROGETTO MIDI

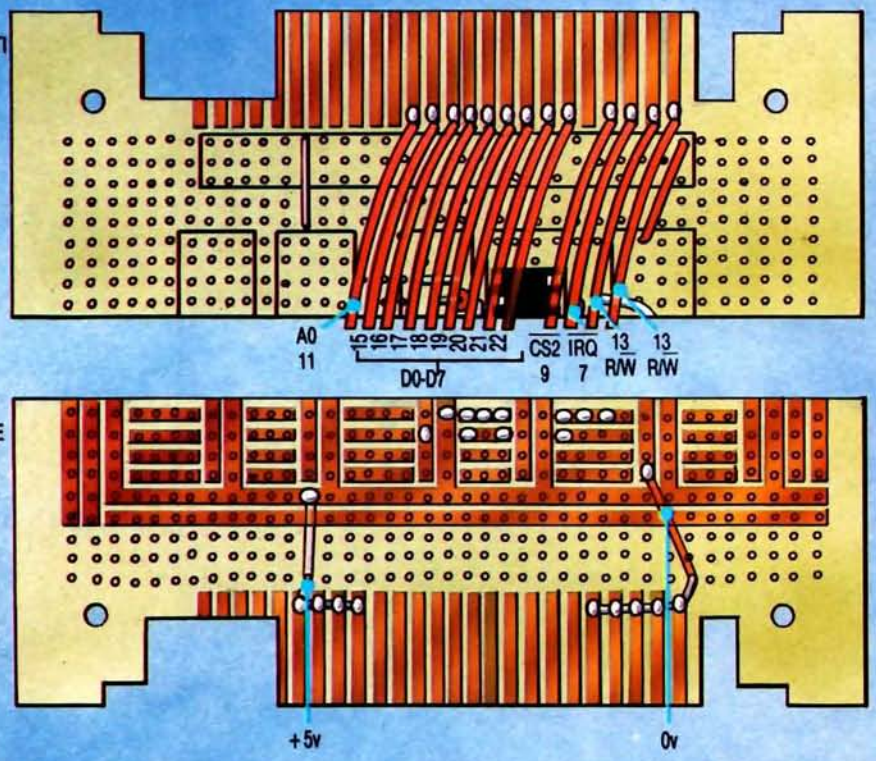
KEVIN JONES

Il bit 2 del registro di stato può generare una interrupt, che non viene utilizzata in questo progetto. Il chip prevede due linee separate per la trasmissione e per la ricezione dei segnali di sincronizzazione (clock), ma in genere esse vengono collegate tra loro. Il clock che regola la frequenza di baud è generato dal segnale di clock in ingresso (che può essere anche diviso per 16 o 32, a seconda del valore dei bit di controllo 0 e 1). I bit 2, 3 e 4 del registro di controllo fissano, rispettivamente: il numero di bit di stop, il numero di bit nei dati e stabiliscono se la parità è pari, dispari oppure se non è prevista. Per adattare il circuito allo standard MIDI si deve porre a 1 i bit 2 e 4 e azzerare il bit 3 (8 bit dei dati, nessuna parità, 1 bit di stop). I bit 5 e 6 del registro di controllo regolano il funzionamento della trasmissione. Per quanto riguarda il sistema MIDI, l'azzeramento del bit 6 consente al bit 5 di abilitare le interrupt del trasmettitore

Commodore 64



Micro BBC



COLLEGAMENTI A PETTINE

Gli schemi a fianco illlustrano i collegamenti dei piedini del chip ACIA per il Commodore 64 e il Micro BBC. I numeri assegnati ai cavetti saldati ai connettori corrispondono a quelli dei piedini dell'ACIA. I piedini sono numerati rivolgendo verso l'osservatore la faccia superiore del chip e orientandone la tacca verso l'alto. I piedini della fila di sinistra a partire dal più vicino alla tacca sono numerati da 1 a 12 mentre quelli di destra da 13 a 24, sempre iniziando dal più vicino alla tacca. I collegamenti sono realizzati con cavetti elettrici rivestiti. Terminato il montaggio, innestare la scheda per il Commodore 64 nella porta d'espansione del computer con il lato dei componenti rivolto verso l'alto e accendere l'apparecchio per collaudarne il funzionamento. Per quanto riguarda la versione per il Micro BBC, è necessario realizzare un cavo di collegamento usando un cavo a nastro a 40 canali, lungo circa un metro, una presa IDC a 40 canali e un connettore a pettine a 40 linee

KEVIN JONES

COLLAUDO DELLA SCHEDA

Innestata la scheda, conviene eseguire alcune prove per controllarne il funzionamento. Se una di esse fallisce, si può usare un multimetro per individuare gli eventuali difetti. Se lo strumento non rivela alcuna anomalia, è necessario effettuare un'accurata ispezione visiva della scheda.

1) Se il computer non funziona normalmente con la scheda inserita:
- verificare che la tensione tra le linee +5v e 0v sia effettivamente di 5 volt. In caso negativo, controllare che tutti gli ICs siano inseriti correttamente e che non vi siano cortocircuiti tra le piste d'alimentazione della scheda
- disinserire la scheda e usare un multimetro per controllare che non vi siano corti circuiti tra i collegamenti sul bus di comunicazione con il computer.

2) Se il computer funziona normalmente, collegare le prese MIDI IN e MIDI OUT dell'interfaccia MIDI usando dei comuni cavetti hi-fi DIN a 5 piedini e digitare il seguente comando (quello tra parentesi tonde è per il Micro BBC):

```
POKE 65032,3 (?&FEE0=3)
```

che carica il valore 3 nel registro di controllo dell'ACIA e produce un reset generale del sistema. L'istruzione:

```
POKE 65032,22 (?&FEE=&16)
```

carica il valore &16 nel registro di controllo e configura l'ACIA nel seguente modo:
- disabilita le interrupt di trasmissione e di ricezione (poiché il sistema non è ancora in grado di gestirle)
- assegna alle parole seriali trasmesse e ricevute una lunghezza di 8 bit (per i dati) più un bit di stop, senza alcuna generazione/verifica di parità
- definisce come frequenza di baud l'impulso di sincronizzazione (clock) ai piedini 3 e 4 diviso per 64 (2MHz/64 = 31,25 KHz, che è la frequenza stabilita dallo standard del sistema MIDI).

A questo punto l'ACIA dovrebbe essere in grado di ricevere e trasmettere dati.
Si verifichi il funzionamento leggendo il registro di stato con l'istruzione:

```
PRINT PEEK (65032) (PRINT ?&FEE0)
```

Il valore letto dovrebbe essere 2. Questo indica che sia il registro di trasmissione dei dati (bit 1 posto a 1) sia il registro di ricezione dei dati (bit 0 posto a 0) sono vuoti. Poiché il computer non ha ricevuto alcun dato e le interrupt sono disabilitate, i bit di stato compresi tra 2 e 7 dovrebbero essere a zero.

3) Trasmettere un byte dal registro di trasmissione a quello di ricezione con il comando:

```
POKE 65033,X (?&FEE1=X)
```

dove X è un numero compreso tra 0 e 255. Questa istruzione carica nel registro dei dati un valore da trasmettere.

4) Il byte viene ricevuto in una frazione del tempo necessario per inserire il successivo comando per la lettura del registro di stato:

```
PRINT PEEK (65032)
(PRINT ?&FEE0)
```

Il nuovo valore dovrebbe essere 3. Il bit 1, azzerato subito dopo l'ultimo comando, viene nuovamente posto a 1 non appena il registro di trasmissione dei dati risulta disponibile per il byte successivo. Il bit 0 viene posto a 1 per indicare che il bit è stato ricevuto e può essere letto dal registro dei dati. In questo caso può verificarsi un errore: se il bit 0 non risulta a 1, probabilmente il circuito del canale di trasmissione non è elettricamente chiuso, quindi l'ingresso del ricevitore rimane a livello alto impedendo la lettura.

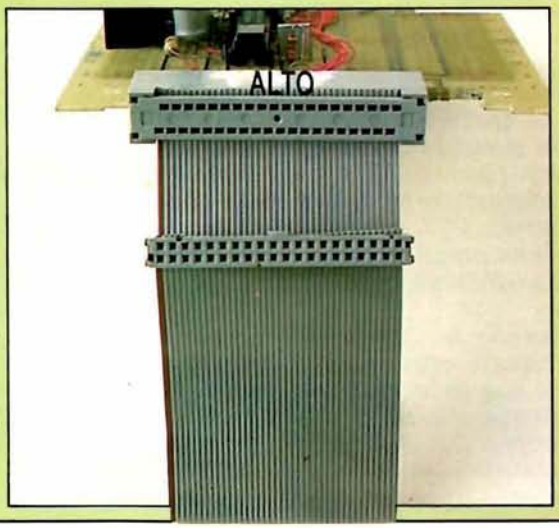
5) Accertata la ricezione di un byte, il seguente comando per la lettura del registro dei dati verifica se il byte ricevuto corrisponde a quello trasmesso:

```
PRINT PEEK (65033) (PRINT ?&FEE1)
```

Questa operazione di lettura dovrebbe fornire lo stesso valore X trasmesso in precedenza. Conviene ripetere le fasi 3-4-5 usando diversi valori per X

IL CAVO A NASTRO

Sul Micro BBC la scheda d'interfaccia MIDI viene collegata direttamente alla porta "tube", situata nella parte posteriore del computer, tramite un cavo a nastro da 40 linee. Alle sue estremità vengono fissati un connettore a pettine IDC a 40 canali e una presa IDC per piedini a 40 linee. Durante l'assemblaggio si faccia attenzione all'orientamento dei due connettori. Collegare il connettore a lama in modo che il piedino 1 (marcato sul connettore) si trovi sulla linea inferiore. Contrassegnare la faccia superiore del connettore per indicarne l'orientamento da usare al momento dell'innesto sulla scheda. Collegare la presa IDC per i piedini all'altra estremità del nastro con la tacca rettangolare rivolta verso l'alto



Il Quick Data Drive

Un'efficiente alternativa alle unità a dischetti della Commodore

Per rispondere alle pressanti richieste degli appassionati di home computer, i fabbricanti hanno prodotto una serie di memorie di massa destinata ai micro più popolari. Tra questi dispositivi c'è il Phonemark 8500 Quick Data Drive, prodotto dalla Dean Electronics per il Commodore 64 e il Vic-20.



CHRIS STEVENS

Un corridore vincente

Il mercato non offre molte alternative alle lente memorie di massa quali le unità a dischetti e a cassette della Commodore. Il Quick Data Drive, comunque è un dispositivo alternativo molto convincente, basato su un sistema "a wafer" che utilizza nastri ad anello identici a quelli del Wafadrive Rotronics. Pur non essendo particolarmente economico, il Quick costa circa la metà dell'unità a dischetti Commodore ma è molto più veloce nella maggior parte delle applicazioni

Questo apparecchio somiglia al Wafadrive della Rotronics (vedere a pag. 1609) per lo Spectrum Sinclair: entrambi adottano drive fabbricati dalla BSR Electronics ed usano "wafer" identici.

Come il registratore a cassette, anche l'unità a dischetti 1541 standard della Commodore è una periferica piuttosto lenta. Ciò dipende dal metodo usato dal computer per caricare i dati. Gran parte del sistema operativo del Commodore 64 deriva dagli apparecchi gestionali PET della metà degli anni '70, che possedevano memorie di massa estremamente inaffidabili, in particolare le unità a cassetta.

Quando la Commodore decise di produrre memorie esterne per i computer PET, optò per una propria unità a cassetta e inserì nel sistema operativo una serie di procedure di verifica che garantissero un corretto caricamento dei dati, riducendo, di conseguenza, la velocità d'accesso alle informazioni. Un sistema simile venne poi applicato al Vic-20 e successivamente al Commodore 64.

Oggi la qualità della registrazione su cassette è notevolmente migliorata e non sono più necessarie procedure di verifica lunghe e complicate. Addirittura molti pacchetti di software gestionale incorporano proprie routine di caricamento che ignorano molte delle precedenti verifiche velocizzando così il caricamento dei dati pur garantendo una notevole affidabilità.

Tuttavia, se in condizioni normali ci si deve ancora adattare alle "lungaggini" imposte dal sistema operativo del Commodore, una risposta c'è: il Quick Data Drive, che riesce ad accedere a programmi e dati con una velocità fino a 15 volte superiore rispetto alle normali cassette e comunque superiore a quella dell'unità a dischetti 1541. L'apparecchio è piuttosto piccolo (circa la metà del registratore a cassette Commodore 1530) e viene collegato al Vic-20 o al Commodore tramite il connettore a pettine per il registratore cassetta.

È previsto un solo drive, mentre il Wafadrive Rotronics ne ha due, ma è possibile collegare al sistema un secondo drive per eseguire elaborazioni che richiedono l'impiego simultaneo di un wafer di sistema e di uno per i dati.

Il sistema operativo

Il fatto che il Quick Data Drive venga collegato alla porta della cassetta non impedisce il contemporaneo impiego di un registratore a cassette.

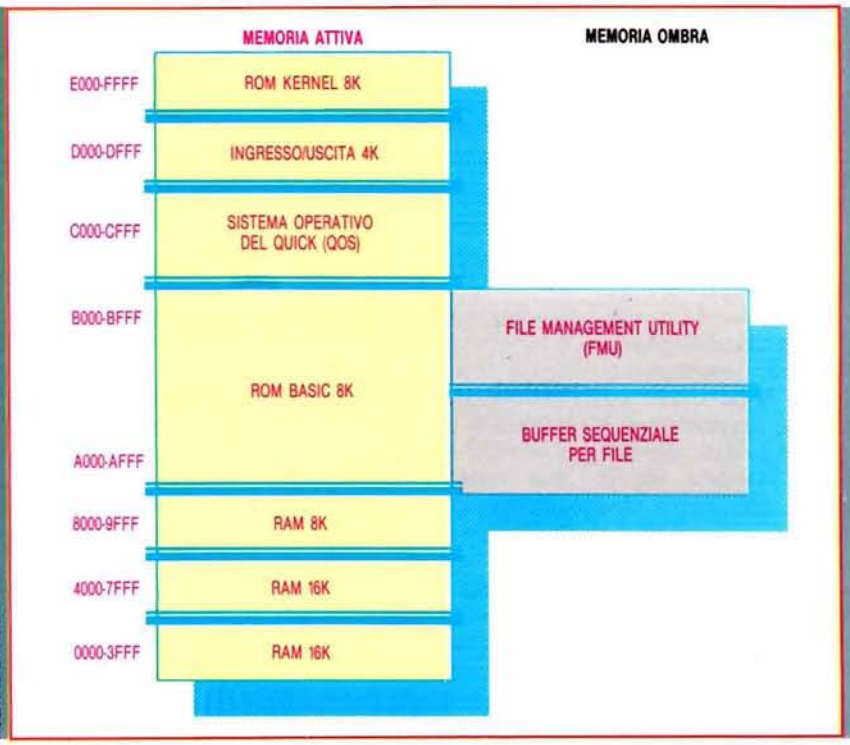
Infatti il Data Drive possiede sul retro un connettore a pettine al quale possono essere collegate "in cascata" altre unità a cassette o un secondo Quick Data Drive.

Nonostante le reciproche somiglianze, il Wafadrive e il Quick Data Drive presentano differenze di funzionamento che derivano dalle diversità degli apparecchi per i quali sono stati progettati. Mentre il sistema operativo del primo è residente in ROM (a imitazione delle ROM per l'Interface 1), quello del Quick Data Drive, chiamato QOS, è registrato su wafer e viene caricato in memoria premendo i tasti SHIFT/RUN (come per un normale caricamento da cassetta). Quando lo schermo visualizza il messaggio PRESS PLAY ON TAPE, l'utente deve premere un piccolo pulsante sul retro dell'unità che serve per inizializzare automaticamente il sistema, permettendo al QOS di eseguire automaticamente le

istruzioni successive. I programmi del QOS vengono caricati in due diverse aree di memoria. Le routine di codice macchina per il caricamento, il salvataggio e la ricerca dei programmi vengono memorizzate tra gli indirizzi C000 e CFFF (nella parte alta della memoria normalmente usata per i programmi in codice macchina). È da questo modulo del QOS che dipende la maggiore velocità di caricamento del drive.
Il QOS non impiega comandi propri in sostituzione di quelli programmati nella ROM del sistema operativo Commodore, esso intercetta le chiamate alle routine che gestiscono le normali procedure di caricamento sostituendo poi ad esse le proprie routine ad alta velocità.
La seconda area di memoria contiene la parte del QOS detta "File Management Utility" (FMU) composta da varie routine. Questa viene caricata tra gli indirizzi B000 e AFFF, che corrispondono ai 4 Kbyte più alti della memoria "ombra" (8 Kbyte), posta sotto la ROM BASIC. I quattro Kbyte più bassi, compresi tra gli indirizzi A000 e AFFF, vengono invece usati come un buffer sequenziale per file da parte della stessa FMU.
Poiché la FMU viene registrata nell'area di memoria posta sotto la ROM BASIC, non è possibile usare contemporaneamente le due zone di memoria.
Pertanto, per accedere alla FMU occorre impartire il comando LOAD 'FMU', che esclude i "banchi" della ROM BASIC e abilita quelli della FMU.
Per dimostrare la maggior velocità di caricamento del Quick Data Drive rispetto al registratore a cassetta, si provi ad esempio a caricare in memoria il gioco di simulazione *Nuovo Mondo* descritto in precedenza (a partire da pag. 1506). Il programma occupa 25 Kbyte ed occorrono ben 9 minuti per caricarlo dalla cassetta, mentre bastano un minuto e mezzo per caricarlo da dischetto e appena 30 secondi per caricarlo dal Quick Data Drive. Comunque, sui sistemi basati su nastri ad anello, il tempo di caricamento dipende essenzialmente dalla posizione della testina di lettura/scrittura rispetto all'inizio del programma.
Il sistema operativo del Quick trova i file di dati richiesti esaminando i blocchi d'intestazione (testate) registrati sul nastro.
Quando il wafer viene formattato, il sistema operativo divide il nastro in blocchi e assegna a ciascuno di essi una sezione per il nome del file.
Ricevuto l'ordine di caricare un file nel computer, il QOS cerca il blocco che contiene la prima parte del file, la carica in memoria quindi trova la seconda parte.
La stessa procedura viene usata per visualizzare la directory del wafer.
Mentre la testina di lettura/scrittura scorre il nastro, il sistema operativo legge tutti i nomi dei file registrando i blocchi che li contengono e quelli vuoti. Infine visualizza una lista dei file e il totale dello spazio disponibile espresso in byte.

Il sistema operativo del Quick

Il SO del Quick Data Drive non è residente in ROM: viene caricato da un wafer di sistema ed è composto da due moduli depositati in due diverse aree di memoria: il QOS (Quick Operating System) viene caricato nell'area solitamente riservata ai programmi in codice macchina, mentre la FMU (File Management Utility) viene registrata nella RAM "in ombra" situata sotto la ROM BASIC, assieme a un buffer nel quale vengono "appoggiati" i file durante la fase di trasferimento verso un'altra unità di memoria



CAROLINE CLAYTON

La "File Management Utility"

Questa parte del sistema operativo contiene una serie di routine controllate da menu per procedure quali la formattazione e la lettura della directory del wafer oppure il trasferimento dei dati da dischetto, da cassetta o da wafer a un wafer d'archivio (di back-up).
Il sistema però non è del tutto privo di inconvenienti. Mentre per i programmi BASIC e i file sequenziali le routine di trasferimento funzionano correttamente, per altri programmi, specialmente quelli in codice macchina da caricare in particolari aree di memoria, possono verificarsi alcuni inconvenienti. Poiché le due aree occupate dal QOS e dalla FMU vengono spesso usate per immagazzinare anche i programmi in codice macchina, il caricamento di questi può "sporcare" (ossia sovrascrivere) il sistema operativo del Quick Data Drive provocando la caduta del programma.
Questa procedura, dunque, consente di copiare i programmi scritti dall'utente ma non sempre il software esistente in commercio. Il problema ovviamente non si presenta se il software è appositamente prodotto e venduto su wafer per il Quick Data Drive, come già è accaduto per il gioco Impossible Mission della Epyx (vedere a pag. 1700).
Da questo punto di vista, il Wafadrive della Rotronics può dimostrarsi un valido alleato; infatti la possibilità di impiegare l'identico supporto sia per il software dello Spectrum sia per quello del Commodore 64; indurrà probabilmente le software house a produrre software a basso prezzo su wafer per entrambi i computer.

QUICK DATA DRIVE

DIMENSIONI

mm. 147 x 118 x 49

INTERFACCE

Presa per il collegamento alla porta per registratore del Vic-20 e del Commodore 64

SUPPORTI MAGNETICI

Nastro ad anello continuo, montato in "wafer"

CAPACITÀ

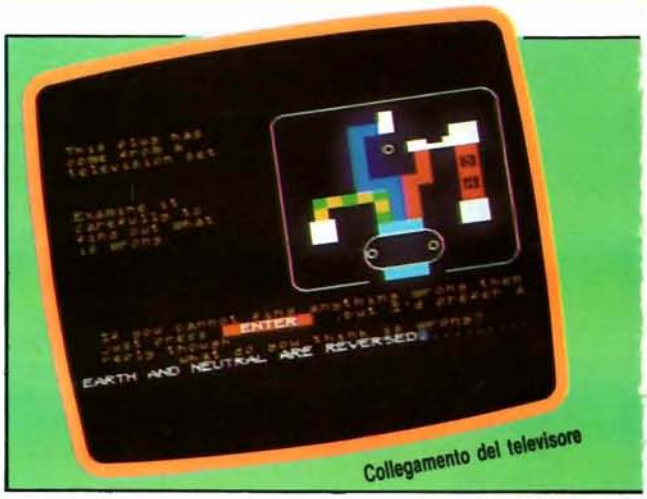
I wafer hanno capacità di 16, 64 e 128 Kbyte

VELOCITÀ DI ACCESSO

Tempi medi di accesso: 8 secondi (file da 15 Kbyte), 43 secondi (file da 120 Kbyte)

Codice che genera codice

Un esame dei vantaggi offerti dai cosiddetti generatori di programmi

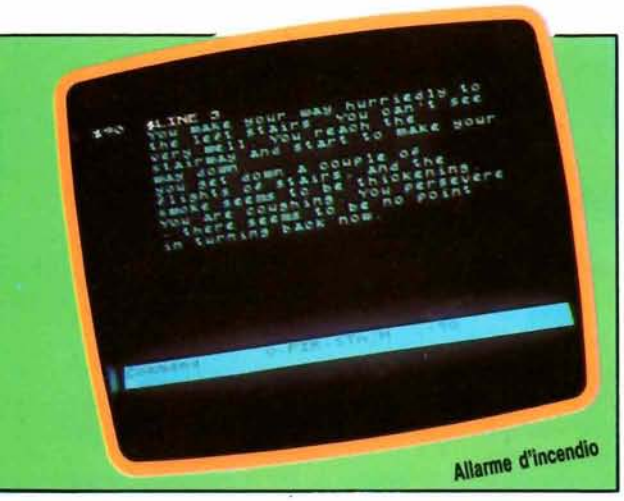


Collegamento del televisore

Il sogno di tutti i programmatori è quello di possedere un programma capace di sostituirli nel faticoso lavoro di programmare e di creare sequenze di codici immediatamente eseguibili che, esenti da errori, non richiedano fastidiose operazioni di debugging.
Per certi versi, chi programma in BASIC usa già un programma "creatore" di programmi: sia residente in ROM, sia caricato da un dischetto, l'interprete BASIC trasforma le linee inserite dall'utente in comandi in codice macchina comprensibili al computer. Questo procedimento è molto più semplice dell'immissione diretta di istruzioni in codice macchina, apparentemente prive di senso, e viene utilizzato anche nella programmazione in linguaggio Assembly per tradurre gli mnemonici Assembly in codice macchina.
Un artificio usato da molti programmatori è quello di creare una routine che genera nuove linee di programma usando sia il buffer della tastiera con un contatore che incrementa i numeri di linea sia inserendole in memoria. Questa tecnica viene abitualmente usata dai programmatori in codice macchina per leggere (PEEK) le locazioni di memoria e introdurle in un programma di caricamento come istruzioni DATA.
Alcuni linguaggi, come il COMAL (vedere a pag. 1602), possiedono un programma interno che intercetta eventuali errori e rifiuta le linee di programma contenenti errori di battitura (ad esempio, PRNT al posto di PRINT). Il FORTH e il LOGO, d'altra parte, non accettano istruzioni che contengano parole non previste dal loro vocabolario o non definite in precedenza dall'utente.
Le tecniche sin qui esaminate non consentono di ordinare al computer la stesura automatica di un programma che calcoli, ad esempio, l'imposta sul reddito, tuttavia la possibilità di sviluppare sistemi che imitino o che possiedano realmente alcune caratteristiche dell'Intelligenza Artificiale rende meno remota tale aspirazione. Ad esempio esiste un programma, chiamato *Microtext*, adatto anche ai meno esperti, progettato per produrre lavori di notevole complessità tecnica, quali programmi didattici interattivi, o per la ricerca di errori oppure per la creazione sia di questionari.sia di software per la ricerca delle informazioni.
I sistemi denominati "d'autore" ("authoring"), abbinati a programmi interattivi per la gestione dello schermo (da usare in fase di verifica), permettono di creare applicazioni software trasformabili in un sistema perfettamente funzionante e modificabile dall'utente. Il *Microtext*, inizialmente prodotto per gli elaboratori CP/M della fascia più alta, è oggi disponibile anche per l'Einstein della Tatung.

I pacchetti "Quill" e "Illustrator"

Un'altro sofisticato generatore di programmi, progettato per creare giochi d'avventura, è il *Quill* (vedere a pag. 522) che permette di sviluppare e modificare un database d'informazioni (contenente indirizzi, oggetti e via dicendo) utilizzabile da parte di un programma Assembly immediatamente eseguibile.
Un secondo pacchetto, dotato di funzioni grafiche, è l'*Illustrator*, che permette di corredare i testi di immagini. Il primo viene prodotto per lo Spectrum Sinclair, l'Amstrad 464 e l'Oric, il secondo per lo Spectrum, l'Amstrad e per gli apparecchi Commodore. Nonostante sia molto economico, il *Quill* è un programma di buon livello ed è stato usato per scrivere molti dei giochi d'avventura oggi in commercio, tra i quali l'*Hampstead* della Molbourne House.
Sebbene entrambi possano essere considerati generatori di programmi, esistono altri due package, molto più completi, largamente impiegati dalle software house: *The Last One* (*TLO*) e il meno potente *Sycero*.
Originariamente il *TLO* è stato scritto per gli apparecchi CP/M a 64 Kbyte (come appare evidente dal suo costante utilizzo di "overlay" (ossia sezioni di programma normalmente residenti su disco e richiamate all'occorrenza), che caratterizzano programmi quale il *WordStar*. Oggi è disponibile anche una versione per computer MS-DOS con una RAM di almeno 256 Kbyte ma, anche se essa è arricchita con nuove sofisticate funzioni, l'esecuzione risulta ancora un po' lenta.
Gli autori del *TLO* affermano che il pacchetto possiede un certo grado di Intelligenza Artificiale, che però non sembra manifestarsi anche agli occhi degli osservatori più accorti. Superficialmente il *TLO* e il *Sycero* si assomi-



Allarme d'incendio

I generatori di programmi

Qui di seguito sono elencati alcuni tra i generatori di programmi più diffusi sul mercato. In molti casi la scelta è fortemente condizionata dal tipo di computer posseduto

"The Last One": Per Apple II e IIe; per apparecchi CP/M-80 e MS-DOS

"Microtext": Per Einstein Tatung e computer CP/M-80

"The Quill": Per Spectrum, Commodore 64, Amstrad 464 e Oric

"Sycero": Per apparecchi MS-DOS

gliano (sono entrambi generatori di programmi) ma la loro struttura interna, sia nelle finalità sia nelle tecniche impiegate per rispondere ai quesiti posti dall'utente allo scopo di ottenere programmi efficienti e "trasferibili", differisce notevolmente. Entrambi possono generare programmi in BASIC e gestire senza particolari problemi le operazioni di immissione, elaborazione, archiviazione, estrazione e stampa delle informazioni. Tuttavia, i programmi risultanti si differenziano da altri, prodotti in modo tradizionale e magari dotati di sofisticate funzioni di database, in quanto essi funzionano indipendentemente dal sistema che li ha generati.
Il *TLO* e il *Sycero* consentono di svolgere gran parte dei calcoli e del lavoro preparatorio alla programmazione sullo schermo, ma questo modo di procedere (che pare essere il preferito dall'utenza attuale) non permette di sfruttare per intero le potenzialità dei due pacchetti. La struttura interna del *Sycero*, articolata nei 7 punti seguenti, mette in rilievo l'importanza di una meticolosa pianificazione preparatoria:

- Progettazione
- Configurazione del sistema
- Creazione delle immagini video
- Verifica dei dati
- Definizione del programma
- Procedure per la stampa
- Generazione del programma

Comunque, nei manuali del *TLO* e del *Sycero* viene sottolineato con insistenza che un buon lavoro di organizzazione e progettazione è essenziale per ottenere buoni programmi.
Nel manuale di *Sycero* si trovano passaggi del tipo:

«L'eliminazione di errori dovuti ad una cattiva progettazione iniziale può richiedere anche più tempo di una riscrittura completa del progetto. La tentazione di soffermarsi soltanto per qualche attimo sulla progettazione e di precipitarsi alla tastiera per passare immediatamente a risultati concreti deve essere evitata nel modo più assoluto...»

«... conviene sempre partire da una completa elencazione delle caratteristiche richieste dal programma finale (ad esempio cosa si desidera far comparire sullo schermo e cosa invece si preferisce stampare). Si passi alla fase operativa soltanto quando si è compilato con precisione tale elenco...»
Il manuale del *TLO* è meno specifico ma altrettanto insistente, con passaggi di questo genere:

«Si prepari un diagramma di massima dell'intera procedura, magari prevenendo le possibilità di errori commessi da utenti inesperti. Per questi ultimi, predisporre "trappole" abbastanza efficaci. L'intero lavoro è facilitato se all'inizio si fraziona tutto il progetto in più programmi di breve lunghezza, collegati tra loro da menu e sottomenu. A lungo andare si trova che questo approccio modulare consente di realizzare anche programmi di notevole complessità...»

Non si pensi, comunque, che questi due pacchetti mettano in grado di generare programmi di qualsiasi tipo. Ad esempio, non è possibile produrre giochi (nemmeno quelli d'avventura basati solo su testi), né word processor o programmi di lavagna elettronica (spreadsheet). La caratteristica più significativa del *TLO* e del *Sycero* è che forniscono all'utente strumenti per manipolare qualsiasi tipo di dati, eseguire calcoli su di essi (dalla semplice scorporazione dell'IVA da un prezzo a lunghe e complesse equazioni utili in ingegneria), memorizzarli, richiamarli e stamparli su carta.
Il lavoro svolto dai due pacchetti è simile a quello di un semplice programma per la gestione di database quale il *dBase II* (vedere alle pagg. 1632 e 1666), ma il loro impiego risulta più semplice ed immediato. Il funzionamento e le funzioni di controllo del *TLO* e del *Sycero* vengono descritte più avanti.

Due esempi applicativi

Queste immagini sono tratte da alcuni programmi creati tramite il "Microtext". Il primo programma (a sinistra) serve per verificare il corretto allacciamento di un televisore alla presa di corrente e rivolge all'utente domande sulle connessioni elettriche. Il secondo (a destra) propone una serie di domande sulle azioni da intraprendere nel caso di un incendio all'interno di un albergo. Entrambi i programmi si basano su procedure interattive e confrontano le risposte dell'utente con il database di conoscenza registrato nella memoria del computer. Questo tipo di applicazioni è il più congeniale ai generatori di programmi.

Interrupt e utili routine

La programmazione in codice macchina e l'Interface 1 offrono notevoli possibilità

Le funzioni controllate da interrupt, nel sistema operativo dello Spectrum sono leggermente più complesse di quelle del Micro BBC (vedere a pag. 1557) e ciò impone una maggiore attenzione nel loro impiego. Se le interrupt vengono inavvertitamente disabilitate, il funzionamento del computer può essere temporaneamente compromesso. Ad esempio, può cessare la lettura della tastiera o verificarsi una caduta del sistema durante l'esecuzione dei programmi BASIC.

La CPU Z80 dello Spectrum risponde a due tipi di interrupt: le interrupt "mascherabili" e quelle "non mascherabili" (NMI). La differenza tra queste è abbastanza semplice: la CPU può essere programmata per ignorare un segnale d'interruzione mascherabile, ma deve sempre rispondere a un'interrupt non mascherabile.

Sullo Spectrum, l'uso delle interrupt NMI può creare inconvenienti, poiché la routine ROM che le gestisce sembra essere difettosa. Nelle intenzioni dei progettisti del SO c'era quella di permettere all'utente la definizione di un indirizzo, nella variabile di sistema situata alle locazioni &5CB0 e &5CB1, dove trasferire il controllo ogni volta che la CPU riceve un segnale NMI. In realtà lo Z80 risponde eseguendo il codice macchina a partire dall'indirizzo &66, provocando così un completo azzeramento del sistema.

Il criterio con il quale la CPU reagisce alle interrupt mascherabili, invece, è piuttosto differente. Queste risposte alternative sono dette "modi di funzionamento dell'interrupt", ma per il momento conviene limitarsi ai modi che hanno significato per il sistema operativo dello Spectrum.

All'inizializzazione del sistema (ossia quando l'apparecchio viene acceso oppure quando l'utente impartisce il comando NEW), la CPU dello Spectrum seleziona automaticamente il modo d'interruzione IM1, mantenuto poi durante il normale funzionamento del computer. L'ULA (Uncommitted Logic Array) del Sinclair invia alla CPU 50 impulsi d'interruzione al secondo. Trovandosi nel modo IM1, la CPU esegue un'istruzione RST &0038 ogni volta che riceve un tale segnale d'interruzione, trasferendo così il controllo alle routine che leggono la tastiera e che aggiornano il contatore FRAMES, situato nelle locazioni comprese tra 23672 e 23674 della RAM. Queste tre locazioni formano un contatore a 24 bit che viene dunque aggiornato ogni 20 millisecondi.

Dopo aver interpretato una linea di programma, l'interprete BASIC attende l'emissione di una nuova interrupt prima di passare all'istruzione successiva. Quindi, se l'utente disabilita le interrupt, l'esecuzione del programma BASIC si ferma.

Dopo l'esecuzione del comando DI (Disable Interrupt), infatti, la CPU ignora le interrupt mascherabili. Per riabilitarle si adopera il comando EI (Enable Interrupt).

Alcune routine del sistema operativo dello Spectrum possono essere eseguite soltanto a condizione che le interrupt siano disabilitate. Tra queste ci sono le routine in cui il tempo gioca un ruolo essenziale (la routine BEEP del generatore di suono e quelle per il caricamento e il salvataggio su nastro, ad esempio). La disabilitazione delle interrupt, inoltre, è temporaneamente necessaria per l'uso della stampante ZX, dell'Interface 1 e del Microdrive. Naturalmente il SO provvede a riabilitare le interrupt al termine della routine.

Durante l'esecuzione di tali routine il contatore FRAMES non viene incrementato: ciò comporta una "pausa d'attesa" nell'elaborazione delle altre operazioni. Mentre la CPU esegue un programma scritto dall'utente, avviene una periodica immissione di interrupt. Se il programma contiene sezioni in codice macchina

PREVEDERE TUTTE LE POSSIBILITÀ

Benché sia ragionevole supporre che il bus dei dati dello Spectrum contenga il normale valore 255, quando viene generata un'interrupt, alcune periferiche (l'interfaccia per joystick della Kempston, ad esempio) possono alterare tale valore.

Per superare questa incertezza, conviene non usare il modo d'interruzione 2 (IM2) se allo Spectrum sono collegate periferiche.

Una tecnica alternativa, che consente la registrazione di qualsiasi valore nel bus dei dati, è quella di riempire una pagina di memoria con valori identici e depositare poi il numero della pagina nel registro I, prima di selezionare il modo d'interruzione IM2. Quando si verifica una interrupt, lo Z80 estrae l'indirizzo della routine di servizio per l'interruzione da una locazione della pagina in questione: l'esatta posizione di questa locazione è determinata dal valore contenuto nel bus dei dati.

Come esempio, si carichi il valore &FC nel registro I e si depositi il valore &FB nei byte compresi tra &FC00 e &FD00. Adesso si selezioni il modo IM2. Se il bus dei dati contiene il valore &C3 quando si verifica un'interrupt, lo Z80 estrae l'indirizzo della ISR dalla locazione &FCC3 (la parte &FC dell'indirizzo viene fornita dal registro I). L'indirizzo ricavato sarà ovviamente &FBFB dato che tutti i 256 byte della pagina FC (più il byte zero della pagina &FD) contengono lo stesso valore.

Questa tecnica si basa su due considerazioni. Innanzi tutto la routine ISR deve sempre trovarsi a un indirizzo composto da un byte basso e un byte alto identici (ad esempio, &C4C4 oppure &FDFD). Secondariamente, occorre sempre ricordare che, quando viene generata una interrupt, il bus dei dati può occasionalmente contenere il valore &FF.

In tal caso lo Z80 cerca la routine ISR agli indirizzi nnFF (byte basso) e (nn + 1)00 (byte alto) dove nn rappresenta il valore del registro I. Per questo motivo è necessario utilizzare anche il primo byte della pagina successiva

che richiedono un'accurata misurazione di intervalli di tempo, è dunque consigliabile disattivare le interrupt all'inizio di tali routine per poi riattivarle prima di tornare al BASIC.

Interrupt vettorizzate

L'uso delle interrupt sullo Z80 (e quindi sullo Spectrum) è più organico se si ricorre al modo d'interruzione IM2, anche se questo è leggermente più complesso del precedente. Mentre l'IM1 produce sempre un salto all'indirizzo &0038 (tramite l'istruzione RST &0038), l'IM2 può essere usato per trasferire il controllo a più routine il cui indirizzo viene specificato usando un "vettore d'interruzione", ossia una tabella di indirizzi in memoria.
In una interrupt vettorizzata, il vettore contiene l'indirizzo della "routine di servizio per l'interruzione" (Interrupt Service Routine o, più brevemente: ISR), che viene eseguita ogni volta che si verifica una interrupt mascherabile. Per conoscere l'indirizzo in memoria del vettore, la CPU ricorre ad un registro speciale, interno allo Z80, chiamato "registro I". L'indirizzo della routine del vettore d'interruzione viene ricavato combinando il contenuto del registro I e quello del bus dei dati al momento dell'emissione dell'interrupt. In alcuni sistemi, la periferica che genera l'interrupt deposita sul bus dei dati un byte che comunica alla CPU la causa dell'interrupt.
Al contrario, l'ULA Sinclair non invia un valore al bus dei dati quando trasmette un segnale d'interruzione allo Z80 perché, grazie alla particolare configurazione dell'hardware dello Spectrum, il bus contiene il valore 255 qualora non venga ricevuto un altro segnale. Quindi il registro I fornisce il byte alto dell'indirizzo a 16 bit del vettore d'interruzione, mentre il valore sul bus dei dati (in questo caso &FF) fornisce il byte basso dell'indirizzo. Per questo motivo il vettore d'interruzione si trova sempre al confine di una pagina di memoria, con il byte basso della routine d'interruzione di servizio registrato all'indirizzo &nnFF e il byte alto all'indirizzo &(nn + 1)00 (dove nn è il contenuto del registro I).
Se, per esempio, il registro contiene il valore &FB, l'indirizzo del vettore si troverà all'indirizzo &FBFF. Il byte basso del vettore, alla locazione &FBFF, conterrà il byte basso dell'indirizzo della routine d'interruzione di servizio, mentre il byte alto del vettore, all'indirizzo &FC00, conterrà il byte alto dell'indirizzo della ISR. La collocazione in memoria della ISR, però, è soggetta a particolari restrizioni. Ad esempio, i primi 16 byte sono riservati alla ROM e non possono essere usati. Inoltre alcuni problemi connessi all'hardware impediscono il corretto funzionamento del SO quando il registro I contiene un valore compreso tra 64 e 127.

Il modo d'interruzione 2

Il seguente programma seleziona il modo d'interruzione 2 dello Z80 e assegna al vettore d'interruzione l'indirizzo della ISR che si intende usare:

```
F3          di          ;disabilita le interrupt
210000      ld   hl,INDIR   ;carica l'indirizzo della ISR in HL
22FFFB      ld   (#FBFF),hl ;assegna l'indirizzo al vettore
3EFB        ld   a,#FB      ;byte alto del vettore...
ED47        ld   i,a        ;... nel registro I
ED5E        IM   2          ;seleziona il modo 2
FB          ei              ;riabilita le interrupt
C9          ret             ;torna al BASIC
```

Naturalmente questo listato presuppone la presenza di una routine per la gestione delle interrupt, situata alla locazione INDIR. Se questa manca, è probabile che il programma (e il computer) si blocchi. È bene che le routine "di servizio per interrupt" scritte dall'utente eseguano tutte le operazioni abitualmente effettuate dallo Spectrum nel modo d'interruzione normale.
Per ottenere questo risultato basta chiamare la routine all'indirizzo &38. Il seguente programma in linguaggio Assembly seleziona il modo d'interruzione IM2 e provoca l'esecuzione della routine alla locazione INDIR ogni volta che riceve una interrupt.

```
            org   60000       ;specifica l'indirizzo di partenza
vettore:    equ   #FEFF       ;indirizzo del vettore
216FEA  cambia: ld    hl,indir    ;carica l'indirizzo in HL
22FFFE          ld    (vettore),hl ;prepara il vettore
F3              di                ;disabilita le interrupt
3EFE            ld    a,#FE       ;preparazione del...
ED47            ld    i,a         ;... registro I
ED5E            im    2           ;cambia il modo d'interruzione
FB              ei                ;riabilita le interrupt
C9              ret               ;ritorna al BASIC
F3      indir:  di                ;disabilita le interrupt
FF              rst   #38         ;normale procedura IM1
FB              ei                ;riabilita le interrupt
C9              ret
```

Chiamando la routine all'indirizzo CAMBIA si seleziona il nuovo modo d'interruzione e si prepara il vettore, dopodiché la routine INDIR viene eseguita ogni cinquantesimo di secondo. Per il momento questa routine non svolge compiti di particolare utilità (a questi si provvede in seguito).
Se, durante l'esecuzione del programma, è necessario tornare al modo d'interruzione normale (IM1), si può usare questa routine:

```
F3          di          ;disabilita le interrupt
3E3F        ld   a,#3f      ;ripristina il valore normale...
ED47        ld   i,a        ;... nel registro I
ED56        im   1          ;modo d'interruzione normale (IM1)
FB          ei              ;riabilita le interrupt
C9          ret
```

Per consentire alla routine di servizio per l'interruzione di eseguire un codice scritto dall'utente, occorre trattare tale codice come una subroutine e chiamarlo con la seguente procedura:

```
F3      indirizzo :di
F5                push af           ;salva i registri
C5                push bc
D5                push de
E5                push hl
FF                rst  #38          ;chiama la ISR normale
F3                di                ;ISR esegue EI, quindi DI
CD0000            call ADD_PROG     ;chiama la routine dell'utente
E1                pop  hl           ;ripristina i registri...
D1                pop  de
C1                pop  bc
F1                pop  af
FB                ei
C9                ret
```

Naturalmente, se la ISR scritta dall'utente è lunga, il funzionamento dello Spectrum viene rallentato, riducendo anche la velocità d'esecuzione del programma e la frequenza con cui viene incrementato il contatore FRAMES. Infine, si rammenti sempre di disabilitare le interrupt durante la routine, per evitare la comparsa di un'ulteriore interrupt quando ancora non è conclusa la procedura associata alla prima.

L'Interface 1 e utili routine

L'Interface 1 possiede una ROM da 8 Kbyte, detta "ROM Ombra", che occupa la prima sezione della memoria e contiene le routine per il controllo delle periferiche collegate (i Microdrive, ad esempio). Inoltre può essere usata per aggiungere all'interprete BASIC comandi quali CAT e FORMAT.
Esitono numerose versioni della ROM Ombra: l'ultima realizzata (Versione 2) è notevolmente migliore della prima (Versione 1), che presentava grosse inefficienze nella gestione dei Microdrive ed era incapace di eseguire molte delle operazioni previste.
Queste inesattezze sono state eliminate nelle versioni successive, che consentono un controllo più affidabile dei Microdrive, permettono di utilizzare la maggior parte dello spazio sulle cartucce e possiedono due nuove funzioni per le stampanti seriali.
Normalmente l'utente accede all'Interface 1 per mezzo del BASIC o di apposite tecniche di programmazione in codice macchina. Nel secondo caso, però, gli indirizzi delle routine nella ROM Ombra dipendono dalla versione in uso. I seguenti esempi si riferiscono sempre agli indirizzi della ROM Versione 1.
Come può la ROM Ombra condividere i primi 8 Kbyte della mappa di memoria con la ROM del BASIC? Tale coesistenza è resa possibile da una tecnica simile al sistema della "ROM impaginata" del Micro BBC. Se l'Interface 1 è collegata, quando lo Z80 tenta di leggere un'istruzione situata agli indirizzi &08 o &1708, la ROM del BASIC viene momentaneamente ignorata e le operazioni successive vengono controllate dalla ROM Ombra.
Al momento opportuno una routine della ROM Ombra seleziona nuovamente la ROM del BASIC.
Naturalmente tutto ciò può verificarsi solo dopo che è stata inizializzata l'Interface 1 e ciò avviene quando:

1) viene digitato il comando NEW con l'interfaccia collegata

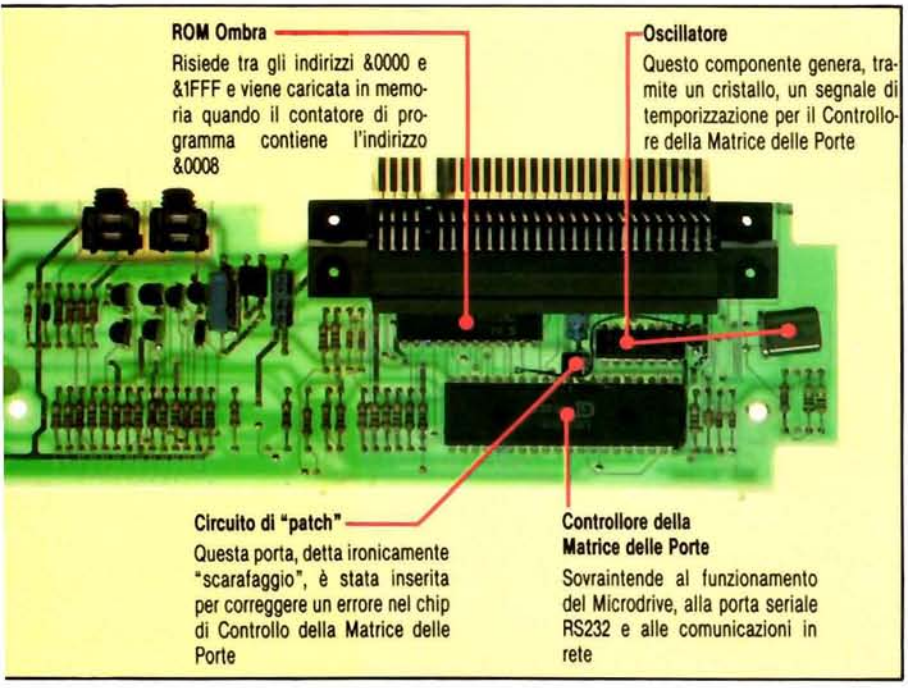
2) la ROM Ombra viene impaginata per la prima volta.

Questa inizializzazione provoca la creazione di un nuovo gruppo di variabili di sistema, che vanno a occupare circa 50 byte di RAM compresi tra la fine delle vecchie variabili di sistema e l'inizio dell'area riservata ai canali. Inoltre, in memoria viene caricata la mappa del Microdrive e altre informazioni relative alle diverse periferiche collegate all'Interface 1. Questa occupazione relativamente massiccia di memoria da parte del sistema spiega perché, quando l'Interface 1 è collegata, si ha uno spostamento verso la parte alta della memoria dell'inizio del testo del programma BASIC. Di conseguenza, se si prevede di usare l'Interface 1, non è consigliabile depositare programmi in codice macchina in istruzioni REM nella linea 1 di programmi BASIC, poiché tale linea non occupa più una posizione costante in memoria.
Come già accennato in precedenza, l'Interface 1 aggiunge tutta una serie di comandi al BASIC, ma permette all'utente di crearne anche di nuovi. Per il momento, però, la descrizione è limitata ai comandi CAT e FORMAT. Sullo Spectrum non espanso, questi comandi non vengono riconosciuti dall'interprete BASIC e generano un errore. La chiamata alla routine d'errore è ottenuta con un'istruzione RST &08, che accede all'indirizzo &08. Tuttavia, quando al sistema è collegata l'Interface 1, la RST impagina la ROM Ombra, che esegue una seconda verifica sul comando non riconosciuto, nel tentativo di interpretarlo correttamente.
Tutti i comandi simili al seguente, che salva sul Microdrive il programma "Testo", funzionano ricorrendo a questa tecnica di riconoscimento.

```
SAVE *"m";1;"Testo"
```

Il simbolo * genera una condizione d'errore che causa il trasferimento del controllo all'indirizzo &08 e, da questo, alla ROM Ombra dove ha luogo l'interpretazione del comando.
Un'ulteriore comodità offerta dall'Interface 1 è che essa mette a disposizione un gruppo di utili routine in codice macchina. Esaminiamo

La scheda dell'Interface 1
La figura mostra i principali componenti dell'Interface 1, tra i quali l'interfaccia seriale e il connettore per il Microdrive. Oltre a fornire le funzioni LAN, l'Interface 1 consente all'utente di creare nuovi comandi BASIC



quelle che si possono considerare di impiego universale, ossia le routine della ROM Ombra che non riguardano il controllo del Microdrive, né l'interfaccia seriale, né le funzioni LAN.

I codici di "aggancio"

Il primo passo riguarda come eseguire la chiamata di queste routine. La ROM Ombra è abilitata soltanto dopo che è stato chiamato uno dei due indirizzi ricordati prima (&08 e &1708). Per chiamare l'indirizzo &08 è sufficiente usare le seguenti istruzioni:

```
RST     &08
DEFB    nn
```

dove nn è un valore compreso tra &1B e &32 (valori esterni a questa gamma generano messaggi d'errore). Tali valori sono detti "codici di aggancio". Ciascuno di essi si riferisce a una particolare routine della ROM Ombra e produce risultati identici, indipendentemente dalla versione della ROM Ombra. L'uso dei codici di aggancio implica alcuni accorgimenti.

1) Poiché, in genere, le routine corrispondenti a questi codici alterano il contenuto dei registri, occorre salvare i registri della CPU il cui contenuto è importante ai fini del programma, oltre alla coppia di registri HL, indispensabili per un corretto rientro nel BASIC.

2) Alcune routine disabilitano le interrupt, quindi è consigliabile riabilitarle subito dopo la chiamata.

3) Prima di chiamare una di queste routine si depositi il valore &5C3A nella coppia dei registri IY.

4) Prima di una chiamata conviene verificare che siano state create le variabili di sistema dell'Interface 1. Per effettuare tale controllo esiste un apposito codice di aggancio:

- *Codice di aggancio 49* (&31): è responsabile della creazione o inserimento delle variabili di sistema relative all'Interface 1. In caso di dubbio conviene eseguire la seguente chiamata, prima di usare qualsiasi altro codice di aggancio:

```
RST     8
DEFB    49
```

Quando la ROM Ombra ha eseguito la routine associata al codice di aggancio, il controllo è restituito alla ROM principale.
La ROM Ombra possiede alcune routine (già presenti nella ROM BASIC anche se in forma meno sofisticata). Ecco quelle che gestiscono l'ingresso da tastiera e l'uscita su schermo:

- *Codice di aggancio 27* (&1B): la routine attende che venga inserito un carattere da tastiera e carica il codice corrispondente nel registro A senza visualizzare il carattere sullo schermo.

- *Codice di aggancio 32* (&20): la routine esegue un'immediata scansione della tastiera e usa il flag di carry per indicare se è stato premuto un tasto (senza tuttavia attendere che ciò accada).
  Se un tasto risulta premuto, il flag C viene forzato a 1 altrimenti è azzerato. Quando vengono chiamati i codici 27 e 32, le interrupt devono essere abilitate poiché la scansione della tastiera, sullo Spectrum, è pilotata dalle interrupt.

- *Codice di aggancio 28* (&1C): la routine invia un carattere (corrispondente al codice ASCII caricato nel registro A quando la routine viene chiamata) al flusso 2, che lo visualizza nella parte superiore dello schermo (vedere a pag. 1812).
  Il seguente programma in codice macchina mostra come vengano impiegati i codici di chiamata 27 e 28 (tuttavia viene innescato un ciclo senza fine che può essere interrotto solo spegnendo il computer).

```
        ;attende un carattere, lo trasmette al flusso 2
        ;Codice di aggancio =1B e =1C
CF      inizio: rst   =08           ;crea le variabili di...
31              defb  =31           ;... sistema dell'Interface 1
FB      ciclo:  ei                  ;tanto per sicurezza
CF              rst   =08           ;attende un carattere...
1B              defb  =1B           ;... dalla tastiera
CF              rst   =08           ;il carattere viene caricato nel...
1C              defb  =1C           ;... reg. A e successivamente stampato
18F9            jr    ciclo         ;il ciclo prosegue all'infinito
```

- *Codice di aggancio 31* (&1F): è simile al codice 28, ma invia il carattere al flusso 3 normalmente usato per la stampante ZX.

- *Codice di aggancio 50* (&32): permette di chiamare una routine della ROM Ombra, adoperandone l'indirizzo dall'interno della ROM BASIC (senza quindi ricorrere ai codici di aggancio).
  Poiché esistono più versioni della ROM Ombra, i programmi che adoperano questo tipo di chiamata possono non funzionare su tutti gli apparecchi.

L'indirizzo della routine che si desidera chiamare viene inserito in una delle variabili di sistema create dall'Interface 1, il cui indirizzo si trova alle locazioni 23789 e 23790.
Il seguente programma chiama la routine specificata:

```
        ;chiamata a routine della ROM Ombra - usare con attenzione!
210000          ld    hl,indirizzo      ;carica l'indirizzo della routine
22ED5C          ld    (23789),hl        ;lo deposita nella variabile di sistema
CF              rst   =08
32              defb  =32               ;esegue la chiamata
```

Questa chiamata, che si dimostra assai utile soltanto se l'utente conosce la versione della ROM Ombra in uso e il funzionamento delle sue routine, viene esaminata più dettagliatamente in seguito, assieme alle modalità d'impiego dell'Interface 1 per la creazione di nuove istruzioni BASIC.

PROGRAMMA MUSICALE CONTROLLATO DA INTERRUPT

Questo programma usa le interrupt per generare un "accompagnamento musicale" sullo Spectrum.
Il "Compilatore dei dati per le note" è un programma BASIC che permette all'utente di comporre accordi, mentre il listato Assembly (o il Caricatore del BASIC, se non si possiede un assemblatore) consente di creare una routine musicale controllata da interrupt e di fornirle dati prodotti dal Compilatore dei dati per le note.
Quest'ultimo programma visualizza un menu a tre scelte: l'opzione C converte l'accordo (contenuto nelle istruzioni DATA 10-900) in un blocco di programma utilizzabile dalla routine d'interruzione. L'opzione P suona l'accordo consentendo all'utente di operare eventuali modifiche. L'opzione R restituisce il controllo al BASIC e permette di salvare i dati compilati o di modificare l'informazione musicale contenuta nelle istruzioni data.
Se viene selezionata l'opzione C, il programma chiede all'utente di specificare l'area di memoria in cui vuole registrare i dati per le note. Questa deve trovarsi al di sopra della RAMTOP, il cui valore va modificato in precedenza per creare lo spazio necessario ai dati. Dopo la compilazione e la registrazione in memoria delle note, il programma visualizza l'indirizzo di base e la lunghezza del codice (queste informazioni vengono utilizzate in seguito). Per salvare il codice, occorre tornare in BASIC e digitare il comando: SAVE "DATI-NOTE" CODE (indirizzo di base), (num. dei byte). Per modificare il motivo musicale, basta inserire nuovi dati nelle linee da 10 a 900.
Per usare il listato Assembly occorre un assemblatore, salvando su nastro il programma generato. In alternativa, si usi il caricatore BASIC.
L'indirizzo di base della routine è 65021 quindi, prima di caricare in memoria il codice, ci si assicuri di avere adeguatamente abbassato la RAMTOP (con il comando CLEAR 65000). Dopo aver assemblato e caricato il codice, si esegua il caricamento del codice prodotto dal Compilatore dei dati per le note.
Per funzionare, il programma musicale deve conoscere in precedenza l'esatta collocazione dei dati relativi alle note da suonare. Questa informazione è fornita alla routine dal seguente programma BASIC:

```
10 LET L=65152: LET V=(*indirizzo
   di base dei dati per le note*)
20 POKE L+1,INT(V/256): POKE
   L,V-PEEK(L+1)*256
```

Infine, l'istruzione RAND USR 65041 seleziona il modo IM2 e inizia a suonare il motivetto. RAND USR 65071 torna a selezionare il modo IM1 e interrompe l'emissione della musica

Compilatore dei dati per le note

```
  1 REM >>> MUSICA CONTROLLATA DA INTERRUPT <<<
  2 REM >> Compilatore dei dati per le note <<
  3 REM
  6 REM  * DATI  PER  LE NOTE  ESPRESSI  NEL
         FORMATO STANDARD PER SPECTRUM (BEEP) *
  7 REM
  9 RESTORE
 10 DATA 1,12,2,9,1,9,1,9,1,8,1,9,2,17,1,12,2,
    12,1,9,2,10,1,10,2,10,1,12,5,14
 20 DATA 1,14,2,7,1,7,1,7,1,6,1,7,2,16,1,14,2,
    14,1,10,2,9,1,9,2,9,1,10,5,12
 30 DATA 1,12,2,9,1,9,1,9,1,8,1,9,2,17,1,12,2,
    12,1,12,2,11,1,19,2,19,1,19,5,19
 40 DATA 1,17,2,16,1,19,1,19,1,18,1,19,2,14,1,
    19,1,19,1,18,1,19,2,12,1,11,2,12,1,11,3,12
 50 DATA 3,10,1,9,1,8,1,9,2,14,1,12,3.2,9,3.2,
    5,3.2,2,3.2,7,5,5
 60 DATA 1,5,1,7,1,9,1,10,2,16,1,14,3.2,12,3.2,
    17,3.2,16,3.2,14,5,12
 70 DATA 1,12,2,14,1,14,1,14,1,13,1,14,3,16,3,9
 80 DATA 2,17,1,17,1,19,1,17,1,19,5,21
 90 DATA 1,21,2,19,1,17,2,14,1,10,3.2,9,3.2,5,
    3.2,7,3.2,4,5,5
997 REM
998 REM *NON CANCELLARE LA LINEA 999*
999 DATA 255,255
1000 REM
2000 **** MENU ****
2010 BORDER 1: PAPER 1: INK 6
2020 CLS
2030 PRINT  AT  5,4;"COMPILATORE MUSICALE  ";AT
     10,5;"C  -- Compila  i  dati  per le note";AT
     12,5;"P  -- Suona  un accordo (con beep)";AT
     14,5;"R -- Rinvia al BASIC" 2040 LET
     a$=INKEY$
2050 IF a$="c" OR a$="C" THEN GO TO 5000
2060 IF a$="p" OR a$ "P" THEN GO TO 3000
2070 IF a$="r" OR a$="R" THEN CLS : STOP
2080 GO TO 2040
3000 RESTORE
3010 READ d,f: IF d=255 AND f=255 THEN GO
     TO 2000
3020 BEEP d/10,f-6: GO TO 3010
5000 RESTORE:  CLS:  INPUT "INDIRIZZO DI BASE DEI
     DATI PER LE NOTE?"; dd1: POKE 23301,
     INT(dd1/256): POKE 23300, dd1-(256*PEEK 23301):
     CLEAR dd1-1: LET dd1 = PEEK 23300+256*PEEK
     23301
5005 LET f=0: LET d=dd1
5010 READ ritardo, altezza
5015 LET ritardo=INT (ritardo*6)
5020 LET freq=(1.0594631^altezza)*256
5030 LET bip=INT ((437500/freq)-30.125)
5035 IF ritardo=255*6 THEN POKE d,255: GO TO 6000
5037 LET f=f+1: PRINT AT 10,10;"NOTA )";f
5040 POKE d,ritardo
5060 POKE d+1, bip-(INT(bip/256)*256)
5070 POKE d+2, INT (bip/256)
5080 LET d=d+3
5090 GO TO 5010
6000 PAUSE 50: CLS: LET lung=(d+2)-dd1
6010 PRINT "INDIRIZZO IN MEMORIA DEI DATI PER LE
     NOTE =";  dd1'''"NUM. DI BYTE DEI DATI PER
     LE NOTE...";  lung 6020 INPUT "PREMERE 'ENTER'
     PER TORNARE AL MENU ";LINE a$: GO TO 2000
```

Caricatore BASIC

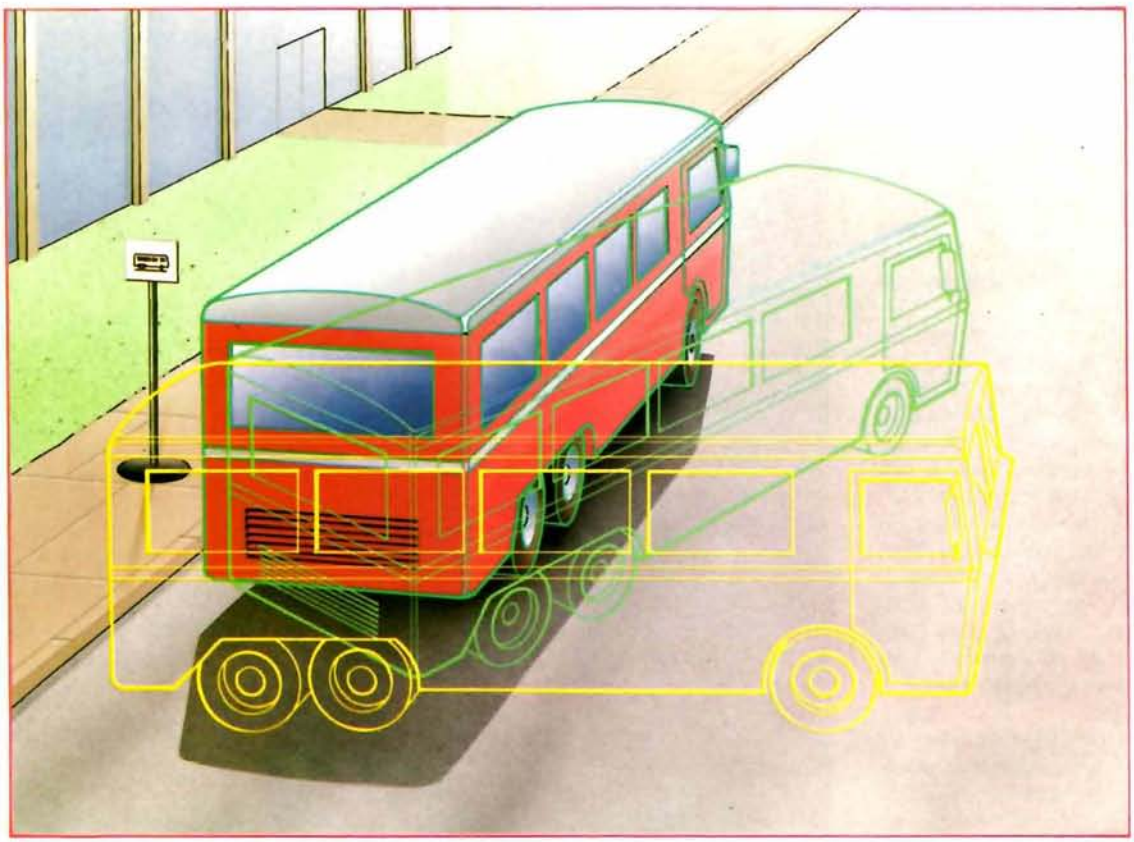
```
 10 RESTORE 9000
 20 CLEAR 64000
 30 LET cqs=0: FOR f=65021 TO 65156
 40 READ dat: POKE f,dat
 50 LET cqs=cqs+dat
 60 NEXT f
 70 IF cqs<>18850 THEN PRINT "ERRORE NEI DATI !!
    !": STOP
 80 PRINT "DATI CORRETTI !"
 90 STOP
9000 DATA 0,0,221,229,229,197,213,245,195,52,254,
     241,209,193,255,221,225,195,56,0,33,255,253,
     34,253,253,243,62,253,237,71,237,94,42,128,
     254,34,126,254,62,1,50,130,254,61,50,131,254,
     251,201,243,237,86,251,101,58,130,254,254,0,
     202,8,254,58,131,254,254,0,202,75,254,61,50,
     131,254,195,8,254,42,126,254,126,254,255,202,
     108,254,50,131,254,35,126,35,95,126,35,34,126,
     254,103,107,17,4,0,205,181,3,243,195,8,254,42,
     128,254,34,126,254,62,1,50,130,254,61,50,131,
     254,195,8,254,0,0,0,0,0,0,0,
```

Listato Assembly

```
                org 65021
0000    vett:   defw 0          ;2 byte per il vettore
DDE5    interr: push ix
E5              push hl
C5              push bc
D5              push de
F5              push af
C334FE          jp              ;salta alla routine musicale
F1      fineprg: pop af
D1               pop de
C1               pop bc
E1               pop hl
DDE1             pop ix
C33800           jp  56         ;salta alla ISR in ROM
21FFFD    on:    ld  hl,interr  ;prepara il vettore
22FDFD           ld  (vett),hl
F3               di
3EFD             ld  a,253
ED47             ld  i,a        ;carica il byte alto di vett in I
ED5E             im  2          ;seleziona il modo IM 2
2A80FE    iniz:  ld  hl,(datind) ;punta ai dati
227EFE           ld  (locaz),hl
3E01             ld  a,1        ;commu = 1 = suona una nota
3282FE           ld  (commu),a
3D               dec a          ;ritardo = 0
3282FE           ld  (ritardo),a
FB               ei
C9               ret            ;ritorna al BASIC
F3        off:   di
ED56             im  1          ;seleziona il modo IM 1
FB               ei
                 ret
C9        ;
3A82FE  start:   ld  a,(commu)  ;carica commu in a
FE00             cp  0
CA08FE           jp  z,fineprg  ;se commu=0 allora fine
3A83FE           ld  a,(ritardo)
FE00             cp  0
CA4BFE           jp  z,nwnot    ;se ritardo=0 allora suona
3D               dec a          ;ritardo=ritardo-1
3283FE           ld  (ritardo),a ;registra il nuovo valore di ritardo
C308FE           jp  fineprg    ;salta alla fine della routine
2A7EFE  nwnot:   ld  hl,(locaz) ;indirizzo del dato in hl
7E               ld  a,(hl)     ;ritardo successivo in a
FEFF             cp  255        ;255=fine dati
CA6CFE           jp  z,azzera   ;dati terminati quindi azzera
3283FE           ld  (ritardo),a ;conserva il nuovo ritardo
23               inc hl         ;punta al succ. byte dei dati
7E               ld  a,(hl)     ;lo copia in a
23               inc hl         ;punta al byte successivo
5F               ld  e,a        ;deposita byte1 in e
7E               ld  a,(hl)     ;deposita byte2 in a
23               inc hl         ;punta al byte successivo
227EFE           ld  (locaz),hl ;conserva l'indirizzo del byte
67               ld  h,a        ;deposita i dati sulla frequenza...
6B               ld  l,e        ;... in HL
110400           ld  de,4       ;durata della nota = 4
CDB503           call 949       ;chiamata alla routine beep (ROM)
F3               di             ;la ruotine ROM ha eseguito una ei
C308FE           jp  fineprg    ;salto a fine routine
2A80FE   reset:  ld  hl,(datind) ;inizializza...
227EFE           ld  (locaz),hl ;... i vari puntatori
3E01             ld  a,1
3282FE           ld  (commu),a
3D               dec a
3283FE           ld  (ritardo),a
C308FE           jp  fineprg    ;salto a fine routine
0000     locaz:  defw 0          ;accantonamento di memoria...
0000     datind: defw 0          ;... per i vari puntatori
00       commu:  defb 0
00       ritardo: defb 0
```

Vedere e capire

Un computer "intelligente" deve saper integrare queste due funzioni



KEVIN JONES

Una immagine... calzante
Alcuni sistemi di riconoscimento delle immagini sono basati su una tecnica induttiva, in cui viene eseguita una scansione dello scenario, alla ricerca di una determinata sagoma. Quest'ultima, rappresentata sotto forma di immagine reticolare, viene proiettata sullo schermo con svariati orientamenti finché i vari punti del reticolo combaciano con l'oggetto eventualmente presente in memoria, individuandolo. Ovviamente, in uno scenario tridimensionale come quello qui raffigurato, il numero degli orientamenti dell'immagine reticolare può diventare altissimo, prima di trovare un corretto abbinamento

Si dice che la bellezza di un oggetto è ingigantita dagli occhi di chi lo possiede anche se, in realtà, essa è concepita dal cervello di chi guarda o, più precisamente, da quella complessa catena di processi neurologici che partono da dietro la retina per terminare nella regione occipitale della corteccia cerebrale. Sono proprio questi i fenomeni che i fisiologi tentano di comprendere e gli esperti di robotica di imitare (pagg. 1161 e 1221).

La percezione visiva è una componente talmente indiscindibile dal processo cognitivo che talvolta, al posto di "capisco", si usa dire "vedo". Nei computer, invece, "vedere" e "capire" sono due elementi ben distinti: una cosa è la raccolta di immagini (che normalmente avviene tramite una telecamera) e un'altra è l'interpretazione di tali immagini, al fine di "comprendere" quali informazioni esse forniscano sull'ambiente circostante. L'acquisizione delle immagini è piuttosto semplice: il problema è interpretarle correttamente.

Il processo di trasformazione che, partendo dalle immagini risale ai significati che esse hanno, può esser suddiviso in tre fasi:

1. *Elaborazione delle immagini:* occorre dare la massima definizione alle forme percepite (magari facendo risaltare i contrasti dei contorni). Questa fase è abbastanza semplice.
2. *Riconoscimento di configurazioni:* è necessario poter individuare la presenza o l'assenza di caratteristiche appartenenti a particolari oggetti. Questa fase richiede molto più impegno della prima.
3. *Comprensione dell'immagine:* è la capacità di interpretare correttamente che cosa sta accadendo nel mondo reale. Questo è il problema più arduo da risolvere.

Sebbene quest'ultima fase non sia stata ancora raggiunta, le altre due hanno già prodotto risultati notevoli.

Riconoscimento di configurazioni

Il processo che permette ad un computer di individuare e riconoscere un particolare oggetto è più complesso di quanto possa sembrare. In genere, gli apparecchi in grado di compiere questa funzione almeno in parte, confrontano quanto raccolto dai circuiti di percezione visiva con una serie di modelli precedentemente memorizzati. Una delle applicazioni più elementari è quella per il ricono-

scimento delle lettere dell'alfabeto: il computer estrae dall'immagine digitalizzata un gruppo di caratteristiche (una configurazione di pixel) e il sistema esegue poi un'analisi alla ricerca di particolari forme "conosciute". Per un uomo, il riconoscimento di una linea obliqua, ad esempio in una lettera, avviene per un processo deduttivo immediato, ma un computer può esser tratto in inganno, specie se la sagoma in esame è stata parzialmente modificata durante la digitalizzazione. Comunque, i sistemi basati su questo tipo di interpretazione scalfiscono appena la superficie di ciò che s'intende per "visione".

Nel campo della ricerca si sono distinte due tendenze contrastanti per ciò che riguarda il riconoscimento di immagini da parte di un computer. Una, chiamata "top-down" (ossia "a deduzione") ricorre all'uso di modelli predefiniti. L'altra, detta "bottom-up", (ossia "a induzione") parte invece dall'interpretazione dei singoli dati (vedere pag. 1222).

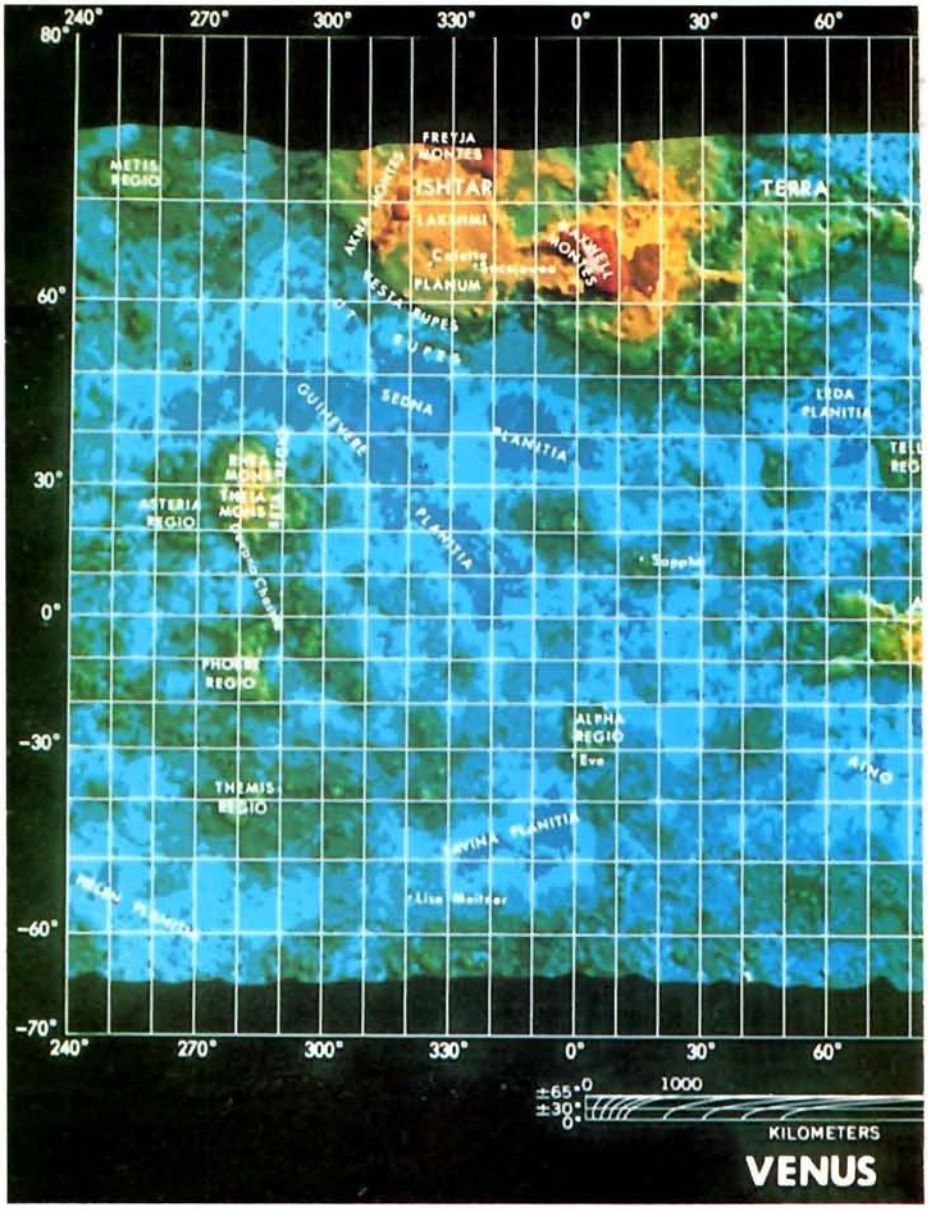
Talvolta l'analisi ottenuta con l'impiego di modelli è chiamata anche "allucinazione controllata": il sistema possiede una definizione (il modello) di un oggetto, la cui proiezione viene fatta ruotare secondo diverse angolazioni e prospettive fino a combaciare il più possibile con l'immagine da riconoscere.

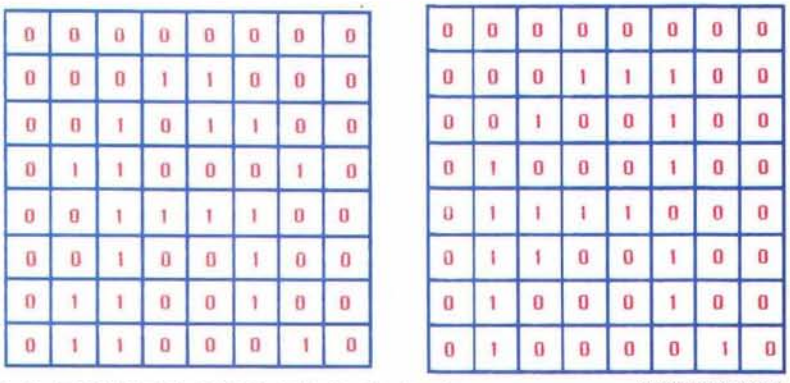
I sistemi "a deduzione", invece, effettuano una scansione metodica dell'immagine, tentando di costruirne una rappresentazione schematica (evidenziando, ad esempio, le linee rette o curve della sagoma). Questo "abbozzo" preliminare viene poi confrontato con configurazioni memorizzate e appartenenti a particolari classi di oggetti.

Nella pratica, i sistemi di riconoscimento delle immagini si avvalgono spesso di una combinazione delle due tecniche appena descritte, anche se ciò è avvenuto solo di recente. In genere tali sistemi ricorrono all'"intelligenza pre-programmata".

Un'eccezione da ricordare è il sistema WISARD (Wilkie, Aleksander & Stonham's Recognition Device) di Igor Aleksander, che può essere istruito a riconoscere una faccia umana sorridente da una imbronciata. Esso è anche in grado di generalizzare la conoscenza a volti mai "visti" prima. Il WISARD lavora in tempo reale (25 "quadri" al secondo) e le sue applicazioni industriali (ad esempio il riconoscimento dei pezzi da scartare fra quelli che scorrono su di un nastro trasportatore) sono forse il preludio ad una più sofisticata generazione di "macchine vedenti".

Molto in breve, il WISARD funziona modificando gruppi di otto pixel alla volta (ottuple) in particolari banchi di RAM. Un'ottupla è un individuatore di caratteristiche e può assumere uno tra 256 valori (da 0 a 255), in base al valore di ciascun pixel della porzione d'immagine considerata. Durante la fase di "apprendimento", nei bit del banco di RAM viene memorizzato 1 se è rilevata la presenza di una particolare sagoma. In seguito, durante la vera fase di riconoscimento, la presenza di un 1 conferma che la prima "impressione" sulla presenza della sagoma era corretta.

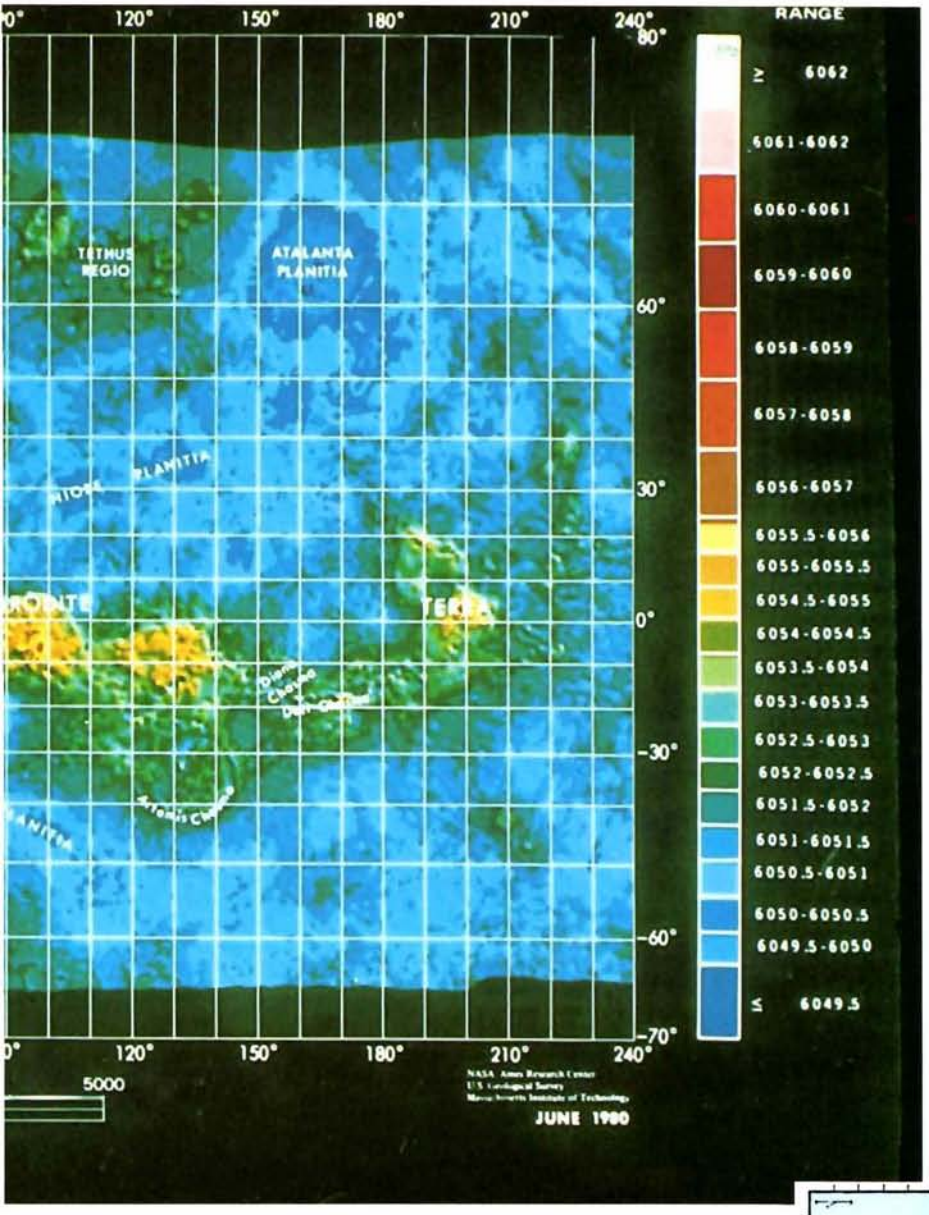




CAROLINE CLAYTON

LA DIFFERENZA TRA R E A

Raramente nel mondo reale esistono dati definiti con estrema precisione. L'esempio qui riportato mostra due rappresentazioni digitalizzate, relative alle lettere A e R. I contorni non risultano ben individuati, a causa di distorsioni e interferenze. I dati cosiddetti "rumorosi" sono il principale problema da affrontare in tutti i sistemi di Intelligenza Artificiale. Quale dei due diagrammi rappresenta la A e quale la R?



Con molte ottuple o "discriminatori", il sistema diventa relativamente insensibile a segnali spuri. Il sistema di Aleksander prevede una griglia di 512 x 512 punti per l'immagine e più di 32000 ottuple. Ciò implica un'occupazione di RAM di circa 8 milioni di bit (un Megabyte). Quantità così notevoli di memoria, ad un prezzo piuttosto economico, sono state rese disponibili solo di recente.

L'ELABORAZIONE DI IMMAGINI

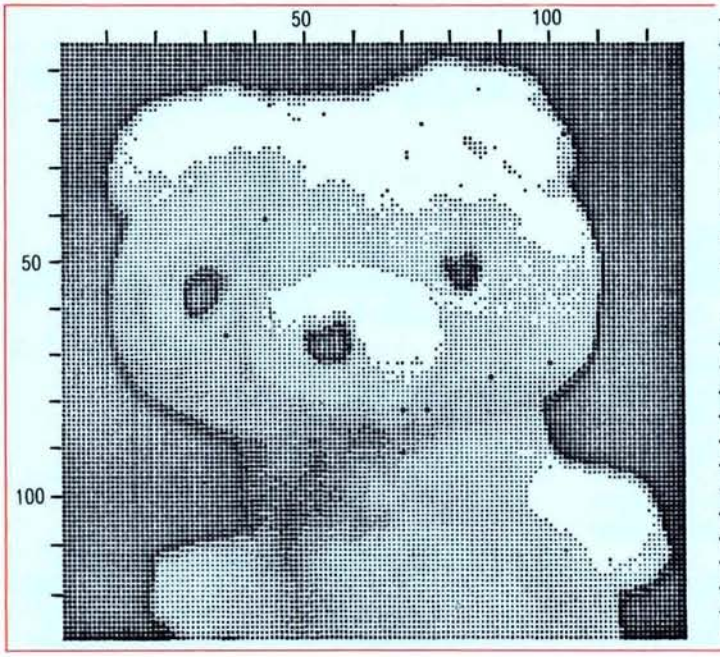
Qui a fianco è mostrato un esempio di elaborazione d'immagine. Nel caso specifico, si tratta di un'immagine rilevata da un satellite e il computer provvede ad accentuarne i contorni, per meglio evidenziare i diversi livelli di elevazione del territorio. Ciò implica l'uso della cosiddetta "falsa colorazione".

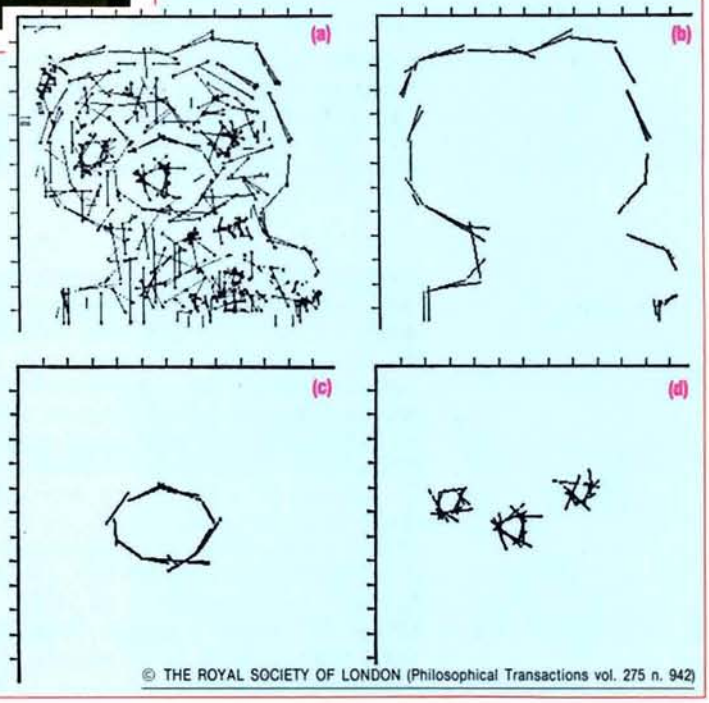
Una particolarità dell'immagine qui riprodotta (una porzione di superficie del pianeta Venere) è che essa è completamente formata da colori "falsi". Questa immagine è visibile soltanto grazie all'elaborazione fatta dal computer; infatti l'occhio umano non è capace di penetrare le dense nubi che circondano il pianeta. Ultimamente, nel campo astronomico, sono state prodotte utilissime (e spesso bellissime) immagini, ricorrendo a lunghezze d'onda cui l'occhio umano non è sensibile (onde radio, raggi infrarossi e ultravioletti). Tuttavia anche in ambienti a noi più vicini vengono adoperate simili tecniche, anche se basate su principi leggermente diversi: si pensi al "sonar", impiegato per costruire immagini del fondo oceanico.

Una buona parte del processo di elaborazione consiste nel tentativo di rimuovere dall'immagine i segnali di interferenza e quelli spuri; i relativi algoritmi sono usati così diffusamente da non venir più considerati come parte dell'Intelligenza Artificiale

"BOZZETTI" PRELIMINARI

Qui sotto è illustrato un metodo di riconoscimento delle immagini non basato sulla conoscenza "a priori" della sagoma analizzata. Il primo "bozzetto" creato dal sistema raccoglie innumerevoli caratteristiche raffigurative dell'oggetto. In seguito (figure b, c e d), vengono estratte alcune caratteristiche salienti per un più preciso riconoscimento dell' oggetto (qui un comune orsacchiotto giocattolo)





© THE ROYAL SOCIETY OF LONDON (Philosophical Transactions vol. 275 n. 942)

Liste e funzioni in LISP

La creazione di liste contenenti liste e l'importanza delle strutture condizionali

In precedenza si è visto come far capire al LISP che una lista deve essere valutata come elenco di dati, anziché come funzione. Così:

```
(SETQ X'(A B C D E F))
```

assegna la lista (A B C D E F), in cui ogni elemento è costituito da una stringa di un solo carattere, alla variabile X. Vediamo che cosa accade se D, E e F sono stringhe di un solo carattere, ma A, B e C sono variabili numeriche il cui valore è, rispettivamente, 2, 4 e 8. In tal caso, si desidera assegnare alla X la lista (2 4 8 D E F). Per fare ciò si ricorre ad una nuova funzione, LIST, che crea una lista di argomenti. Pertanto:

```
(SETQ X (LIST 'A 'B 'C 'D 'E 'F))
```

produce un risultato identico alla prima espressione considerata, mentre con:

```
(SETQ X (LIST A B C 'D 'E 'F))
```

si ottiene l'assegnazione di (2 4 8 D E F): è bastato omettere l'apostrofo nei primi tre elementi. Analogamente:

```
(SETQ X (LIST 1 2 4 '(PLUS 4 4))
```

assegna a X la lista (1 2 4 (PLUS 4 4)) in cui il quarto elemento è di per sé una lista contenente tre elementi: PLUS, 4 e 4. In altre parole, la funzione PLUS non è riconosciuta come tale (e non viene eseguita), a causa dell'apostrofo che precede la parentesi. Al contrario:

```
(SETQ X (LIST 1 2 4 (PLUS 4 4))
```

assegna a X la lista (1 2 4 8), poiché stavolta la funzione PLUS viene regolarmente eseguita. Ciò sottolinea l'importanza da attribuire, in LISP, agli apostrofi. Il concetto può essere ulteriormente esteso per creare una lista di liste. Ad esempio:

```
(SETQ PERSONA '((GIGI ROSSI) (17 1 1960)))
```

assegna alla variabile PERSONA una lista di dati, composta da due elementi, ognuno dei quali è a sua volta una lista: la prima contiene due sequenze di caratteri, la seconda 3 valori numerici. La creazione di liste composte da liste può essere ancora estesa.
Le funzioni fin qui esaminate sono: SETQ, LIST, PLUS e TIMES. Tuttavia ne esistono ancora molte interne al LISP, il cui impiego dipende dalla versione usata. Inoltre, esiste la possibilità di creare funzioni completamente nuove e, in effetti, è proprio ciò che accade quando si scrive un programma in LISP.
Prima di vedere qualche esempio pratico, però, vale la pena di esaminare tre funzioni fondamentali, presenti in tutte le versioni di LISP: CAR, CDR e CONS. I nomi CAR e CDR derivano dalle prime implementazioni del linguaggio (sono l'acronimo di Contents of Address Register, ossia contenuto del registro degli indirizzi e Contents of Decrement Register, ovvero contenuto del registro di decremento). Sia CAR che CDR prevedono un solo argomento, che deve essere una lista composta da almeno un elemento, con la sola eccezione della speciale struttura nota come "lista nulla", o NIL, così rappresentata:

```
()
```

La funzione CAR fornisce come risultato il primo elemento della lista usata come argomento, pertanto:

```
(CAR '(1 2 3 4 5))
```

fornisce come risultato il valore intero 1. Tuttavia, non c'è ragione che impedisca al risultato di essere esso stesso una lista. Per esempio, nell'espressione:

```
(CAR '((1 2) (3 4) 5))
```

il primo elemento della lista è (1 2).
La funzione CDR estrae dalla lista usata come argomento tutti gli elementi eccetto il primo (ossia quanto viene lasciato nella lista originale dopo l'esecuzione di una CAR):

```
(CDR '(1 2 3 4 5))
```

fornisce la lista (2 3 4 5), mentre:

```
(CDR '((1 2) (3 4) 5))
```

fornisce la lista ((3 4) 5).
CONS prevede due argomenti, che provvede a concatenare, "sommando" il primo al secondo. Pertanto:

```
(CONS 1 '(2 3 4 5))
```

costruisce la lista (1 2 3 4 5), mentre con:

```
(CONS '(1 2) '((3 4) 5))
```

si ottiene la lista ((1 2) (3 4) 5).
Spesso nei programmi le funzioni CAR e CDR vengono nidificate, cosicché:

```
(CDR '((1 2) (3 4) 5))
```

fornisce ((3 4) 5), mentre con:

```
(CAR (CDR '((1 2) (3 4) 5)))
```

si ottiene (3 4), che equivale a (CAR '((3 4) 5)), e da:

```
(CAR (CAR (CDR '((1 2) (3 4) 5))))
```

si ha il valore 3, che equivale a (CAR '(3 4)).
Una simile notazione estesa può diventare fastidiosa, pertanto il LISP ammette alcune forme abbreviate. I nomi delle funzioni inizia-

no sempre per C e terminano con R, ma in mezzo è consentita qualsiasi combinazione di A (come in CAR) e di D (come in CDR). Nel LISP della Acornsoft, ad esempio, la combinazione dei caratteri interni può essere lunga fino a tre lettere.
L'ultima espressione può essere riscritta come:

```
(CAAR (CDR '((1 2) (3 4) 5)))
```

oppure:

```
(CAR (CADR '((1 2) (3 4) 5)))
```

o anche semplicemente:

```
(CAADR '((1 2) (3 4) 5))
```

in tutti i casi il risultato è identico: 3.

Definizione di funzioni

I programmi LISP, come già accennato, sono composti in gran parte da funzioni definite dal programmatore. Per la loro creazione si ricorre alla funzione DEFUN, che prevede tre argomenti nella forma:

```
(DEFUN a (b) (c))
```

dove a è il nome della funzione, b è la lista dei parametri (simile a quella del BASIC BBC, del PASCAL, ecc.) e c è una struttura a lista, che contiene il corpo principale della funzione.
Vediamo un esempio pratico. Supponiamo di avere la frequente necessità di moltiplicare valori numerici per la costante 8. Si potrebbe scrivere, per ogni caso, semplicemente:

```
(TIMES 8 N)
```

dove N è il numero da moltiplicare. Tuttavia conviene definire una funzione del tipo:

```
(DEFUN TIME8 (N) (TIMES 8 N))
```

Qui la funzione TIME8 prevede l'argomento N e adopera la normale funzione TIMES per calcolare il risultato. Cosicché l'espressione:

```
(TIME8 11)
```

produce il valore 88.
Prima di procedere oltre con le funzioni, è bene esaminare un importante concetto di programmazione: la condizione. La sua importanza è fondamentale in ogni linguaggio (si pensi a un BASIC privo di espressioni del tipo IF... THEN !)
Le espressioni condizionali, in LISP, hanno un formato simile alle funzioni:

```
(COND (Condizione1 Espressione1)
      (Condizione2 Espressione2)
            :          :
      (CondizioneX EspressioneX))
```

Prima di tutto, si osservi che la funzione si estende su un arco di più righe. Ciò è abbastanza comune in LISP, che riconosce il termine di un'espressione solo quando il numero di parentesi aperte e chiuse coincide.
La funzione COND è molto simile alla CASE del PASCAL (vedere a pag. 1526). Per prima viene valutata la Condizione1 e, se questa è vera, viene eseguita l'Espressione1. In caso contrario si passa alla Condizione2 e così via.
In alcune versioni di LISP (tra le quali quella della Acornsoft), è obbligatorio che vi sia almeno una condizione (o "predicato") che possa risultare vera.
Ciò può essere previsto facilmente anche nei casi più complessi aggiungendo una condizione finale del tipo:

```
(COND (Condizione1 Espressione1)
      (Condizione2 Espressione2)
            :          :
      (T                Espressione finale))
```

Qui il carattere T è usato per indicare "True" (vero) cosicché, in caso falliscano tutte le condizioni precedenti, viene sempre eseguita l'espressione finale. In LISP:

```
T = Vero = Non Zero
```

e:

```
F = Falso = Zero = ()
```

(L'ultima espressione è la lista vuota o "nulla".)
Proviamo adesso a formulare una funzione più complessa, equivalente alla ABS del BASIC. Questa fornisce un risultato che equivale al valore positivo del suo argomento. In altre parole, se l'argomento è positivo esso viene lasciato inalterato, ma se è negativo il suo segno viene cambiato.
La funzione LISP ha questa forma:

```
(DEFUN ABS (X) (corpo della funzione))
```

dove X è l'argomento di tipo intero. Usando la funzione condizionale, la funzione ABS può essere così definita:

```
(DEFUN ABS (X)
   (COND ((MINUSP X) (MINUS X))
         ( T          X        )))
```

Qui vengono impiegate altre due nuove funzioni. MINUSP esegue una verifica sul segno dell'argomento e, se questo è negativo, il risultato è "vero" altrimenti è "falso". Se la condizione risulta "vera", la funzione MINUS cambia segno (ossia "nega") il valore originale.
Se invece il valore originale è positivo, la funzione T (sempre vera) produce il valore X.

```
(DEFUN UGUALE (A B) (COND
   ((EQ A B) T)
   ((OR (ATOM A) (ATOM B)) NIL)
   ((UGUALE (CAR A) (CAR B)) (UGUALE (CDR A) (CDR B)))
   (T NIL)))
(DEFUN TROVA_AUTO (X L) (COND
   ((NULL L) '(NESSUN NUMERO))
   ((UGUALE X (CDAR L)) (CAR L))
   (T (TROVA_AUTO X (CDR L)))))

(SETQ AUTO '((GIULIANI RI762009) (CARBONI MIX93833)
             (ROSSI VE928484) (BIANCHI PA928734)))
(PRINTC (TROVA_AUTO '(VE928484) AUTO))
```



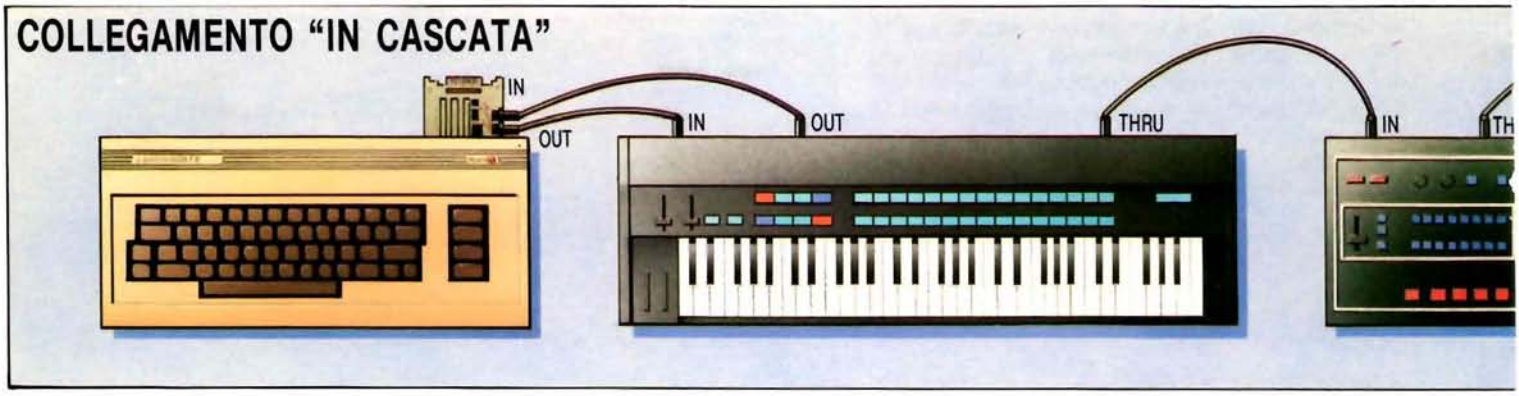
Procedura di ricerca

Questo semplice programma per l'estrazione di dati dimostra l'impiego delle funzioni CDR e EQ. Immettendo il numero di targa di un veicolo, il programma fornisce una lista composta da due elementi (il numero di targa e il cognome del proprietario del veicolo) purché, ovviamente, tali informazioni siano contenute nell'archivio usato dal programma

Funzioni di verifica

Ecco un elenco di funzioni di verifica spesso usate in LISP:

(OR arg1 arg2...)	Produce "vero" se almeno uno degli argomenti è vero
(ONEP arg)	Vero se arg = 1
(ZEROP arg)	Vero se arg = 0
(LISTP arg)	Vero se arg è una lista, falso se è un atomo
(ATOM arg)	Vero se arg è un atomo, falso se è una lista
(LESSP arg1 arg2)	Vero se arg1 < arg2
(GREATERP arg1 arg2)	Vero se arg1 > arg2
(EQ arg1 arg2)	Vero se arg1 = arg2



Passa parola...

Il metodo più diffuso di collegare le varie apparecchiature compatibili con il sistema MIDI è quello chiamato "in cascata", che si avvale in particolare della presa identificata con MIDI THRU. Questa consente una connessione in cui le linee di trasmissione sono comuni a tutti gli strumenti, che vengono selezionati in base a specifici codici di identificazione

Regole di composizione

Realizzata l'interfaccia MIDI esaminiamo il software necessario

Come inviare un byte di dati ad uno strumento, attraverso la MIDI, è già stato spiegato; adesso occorre determinare il formato del byte (o dei byte) occorrenti per comunicare le informazioni necessarie. Un singolo "evento" musicale viene trasmesso sotto forma di "messaggio", ossia come gruppo di byte. La maggior parte dei messaggi è lunga due o tre byte, con l'eccezione dei messaggi "esclusivi di sistema", esaminati più avanti, che possono contenere un qualsiasi numero di byte. Ogni messaggio inizia con un byte in cui il bit più significativo (BPS) è posto a 1. Seguono i restanti byte del messaggio, aventi il BPS posto a 0. I byte che possiedono il BPS pari a 1 vengono chiamati "byte di stato", gli altri semplicemente "byte di dati". I messaggi MIDI sono suddivisi in due categorie principali: quelli "di canale" e quelli "di sistema". La necessità di avere messaggi di canale deriva dall'impiego di collegamenti "in cascata", adottati principalmente nei sistemi più piccoli. In tal caso, ogni unità riceve tutti i segnali emessi dall'unità trasmittente, ma riconosce ed esegue solo quelli relativi al "numero di canale" assegnatole (da 1 a 16). I messaggi di canale sono composti da un byte di stato il cui valore è compreso tra $80 e $EF e da uno o due byte di dati. Il numero di canale è codificato nei quattro bit meno significativi del byte (cifra esadecimale meno significativa) del byte di stato. Pertanto il valore $00 corrisponde al canale 1 e $0F corrisponde al canale 16. I rimanenti tre bit determinano il tipo del messaggio seguente. Una caratteristica speciale dei messaggi di canale è che non occorre ripetere il byte di stato, se questo è identico a quello trasmesso nel messaggio precedente. Pertanto una volta inviato un byte di stato, questo rimane attivo finché non se ne invia uno nuovo (l'unica eccezione a ciò si presenta quando un messaggio di sistema in tempo reale interrompe temporaneamente lo stato "corrente").

Questo criterio di funzionamento è particolarmente comodo se si tratta di inviare consecutivamente più messaggi per l'emissione (on) o l'annullamento (off) di note. L'annullamento di una nota si ottiene specificando una "velocità" pari a zero. La possibilità di mantenere inalterato il byte di stato consente dunque di risparmiare memoria e di velocizzare l'emissione di note.

L'assegnamento della "voce", nei sintetizzatori, è il processo che consente di dirigere un messaggio (sia esso proveniente dal sistema MIDI o da tastiera) a uno dei sintetizzatori collegati al sistema. Se, per esempio, un sintetizzatore polifonico permette la simultanea emissione di sei note si dice che ha sei voci. Per controllare la risposta dello strumento al messaggio inviatogli sono disponibili svariati messaggi modali MIDI. Un messaggio modale, comunque, dovrebbe essere ignorato da uno strumento ricevitore che non sia capace di operare nel modo specificato.

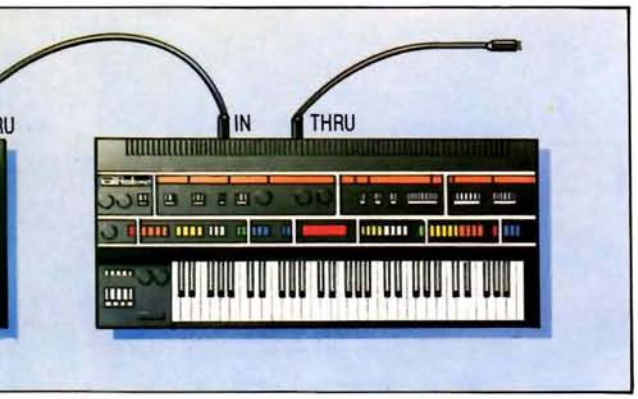
I messaggi modali MIDI

Il modo più semplice è l'omni mode, in cui il ricevitore risponde a tutti i messaggi di canale, qualunque sia il numero di canale specificato nel messaggio. Se l'omni mode viene disattivato, l'unità risponde solo ai messaggi per il canale MIDI ad essa associato.

La maggior parte dei sintetizzatori polifonici ha un numero limitato di voci (di solito sei oppure otto). Ciò significa che occorre prevedere il caso in cui venga richiesta l'emissione di una nota e tutte le voci sono già state impegnate da altri messaggi.

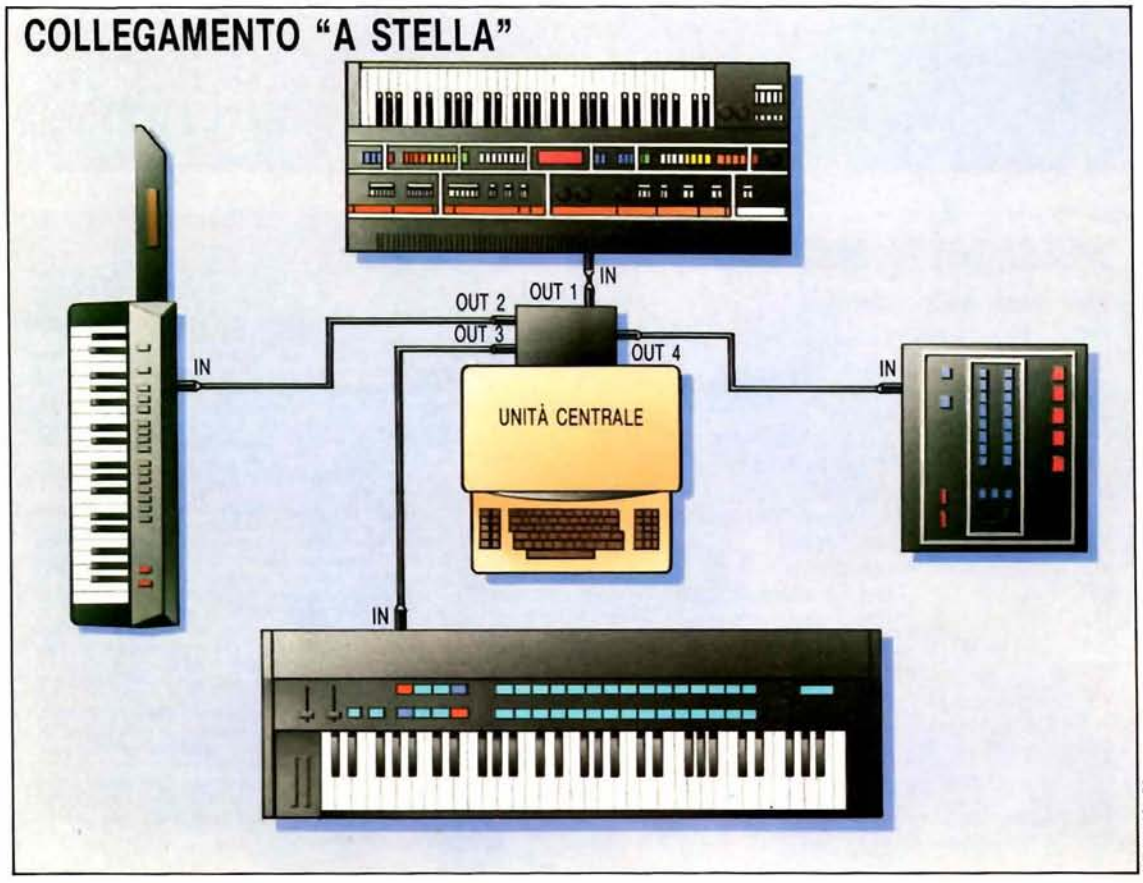
Di solito l'assegnamento delle note avviene con un criterio di rotazione, cosicché la nota "più vecchia" viene annullata per procedere all'emissione della più recente. Questo criterio è selezionato abilitando il modo poly.

L'abilitazione del modo mono, invece, predi-



spone il dispositivo ricevente in maniera da assegnare ciascuna delle sue voci ad uno tra un gruppo di canali MIDI consecutivi, iniziando dal canale principale (o fondamentale) del sintetizzatore. Il messaggio del modo mono viene inviato sul canale principale e il secondo byte di tale messaggio specifica quanti canali sono richiesti.
Se è attivo il modo omni, un messaggio mono istruisce semplicemente il ricevitore ad assegnare una voce monofonica ai messaggi di canale MIDI su tutti i canali.
I *messaggi di sistema* non contengono specifiche di canale nel byte di stato, così il loro invio concerne tutti gli strumenti collegati al sistema. I messaggi si dividono in tre categorie: i messaggi comuni, in tempo reale ed esclusivi.
I *messaggi di sistema comuni* sono composti da un byte di stato (che può assumere valori compresi tra $F1 e $F7) e da 0, 1 o 2 byte di dati.

I *messaggi di sistema in tempo reale* riguardano tutti i componenti del sistema e sono composti soltanto da un byte di stato (di valore compreso tra $F8 e $FF) e nessun byte di dati. Questa particolarità ne permette l'invio in ogni momento, anche interrompendo altri messaggi in corso d'esecuzione.
Principalmente questi messaggi servono per la sincronizzazione di dispositivi quali sintetizzatori di batterie e sequenziatori. La maggior parte dei normali sintetizzatori, comunque, ignora i messaggi di questo tipo, se non possiedono al loro interno un sequenziatore capace di sincronizzarsi con il sistema MIDI.
I *messaggi di sistema esclusivi* iniziano con un byte di stato il cui valore è $F0, seguito da un numero qualsiasi di byte di dati. Il messaggio può terminare con il byte di fine-messaggio $F7 o con un qualsiasi altro byte di stato. Il primo byte di dati è il codice identificativo (ID) del fabbricante: se esso non corrisponde al codice dell'apparecchio ricevente quest'ultimo deve ignorare il messaggio.
I messaggi di questo tipo servono per scambiare informazioni tra strumenti dello stesso genere. I dati trasmessi, in genere, costituiscono particolari sequenze di programmazione ("patch") per sintetizzatori. Questa trasmissione non va confusa con il messaggio di canale $Cx, che opera una selezione tra i programmi patch già residenti nella memoria del sintetizzatore ricevente.
Il contenuto dei messaggi di sistema esclusivi, di solito, compete ai fabbricanti, purché sia stato assegnato loro un codice d'identificazione valido.



Una "star" d'eccezione

Le prime implementazioni del sistema MIDI non prevedevano la selezione tramite numero di canale. Un collegamento "in cascata" non aveva quindi alcun senso, poiché tutti gli strumenti tentavano di rispondere ai messaggi emessi dal trasmettitore. L'unica forma di collegamento utile era quindi quella detta "a stella", al centro della quale si trova l'unità centrale di trasmissione, dotata di un'uscita per ciascuna unità periferica

MESSAGGI PER I CANALI DELLE VOCI

Stato	Dato1	Dato2	Descrizione	Note
$8x	p	v	Nota off p = altezza v = velocità di off	1,2
$9x	p	v	a) v > 0 Nota on p = altezza v = velocità	1,2
			b) v = 0 Nota off p = altezza	1,2
$Ax	p	pr	Post-pressione polifonica p = altezza pr = pressione	1,3a
$Bx	n	c	a) n < $7A Cambio di controllo n = numero del controllore c = valore del controllore	4
			b) n = $7A Controllo locale on (c=$7F) off (c=0)	5
			c) n = $7B, c=0 Tutte le note off	6
			d) n = $7C, c=0 Modo omni off (tutte le note off)	6
			e) n = $7D, c=0 Modo omni on (tutte le note off)	6
			f) n = $7E Modo mono on (tutte le note off) c = numero di canali	6,7
			g) n = $7F, c=0 Modo Poly on (tutte le note off)	6
$Cx	pp	-	Selezione programma "patch" pp = numero del programma	
$Dx	pr	-	Pressione sul canale pr = valore di pressione	3b
$Ex	l	m	Variazione della "ruota" per altezze l = 7 Bit meno significativi m = 7 Bit più significativi	8

Note

Se non altrimenti specificato, i byte dei dati possono assumere un qualsiasi valore compreso tra 0 e $7F. La x rappresenta il numero di canale, che occupa la cifra hex meno significativa del byte di stato (x = 0 corrisponde al canale 1 e x = $0F al canale 16).

1. Il valore di p indica l'altezza della nota in semitoni. Il Do centrale corrisponde a p = $3C e tutti i Do corrispondono a valori multipli di $0C (12 decimale). Il valore di p per una regolare tastiera da pianoforte, con 88 tasti, varia dunque da $15 a $6C.
2. La velocità v può assumere valori compresi tra $01 e $7F. Nelle tastiere prive di sensori di velocità il valore dovrebbe essere $40. Un messaggio di "nota on" con una velocità zero equivale ad un messaggio di "nota off" con una velocità di $40. In genere questo è il metodo adottato con le tastiere più semplici, prive di sensori di rilascio, poiché consente di usufruire dello stato corrente dei messaggi di canale.
3. La "post-pressione" è la quantità di pressione operata sul tasto dopo che questo è stato premuto. Di solito viene impiegata per ottenere una modulazione senza dover intervenire sui parametri di modulazione. Esistono due tipi di sensori di pressione, pertanto sono previsti due diversi tipi di messaggio.
   a) La post-pressione individuale o polifonica implica la presenza di sensori di pressione su ciascun tasto ed ha effetto soltanto sulle note corrispondenti ai tasti premuti. Di conseguenza, oltre al valore della pressione, va inviato anche quello dell'altezza della nota (devono inoltre essere previsti circuiti di modulazione per ogni singola voce).
   b) La pressione di canale è prodotta da un unico sensore di pressione, che influenza in modo identico tutti i tasti. La pressione trasmessa corrisponde alla pressione massima (tutti i tasti premuti simultaneamente). L'effetto risultante riguarda tutte le voci.
4. Questi messaggi sono inviati da unità di controllo esterne alle tastiere. Numeri di controllore compresi tra $0 e $1F corrispondono a controllori "continui" ed hanno un valore di controllo (c) tra $0 e $7F. In linea di principio queste unità di controllo assomigliano ai joystick per computer basati su potenziometri (in pratica possono essere dispositivi quali gli stessi joystick, controlli a pedale o a fiato e via dicendo). I numeri di controllore compresi tra $20 e $3F sono eventualmente adoperati per l'invio di ulteriori sette bit meno significativi destinati ai controllori da $0 a $1F, qualora occorra un'altissima definizione. I numeri di controllore compresi tra $40 e $5F corrispondono a unità di controllo basate su interruttori (quali pedali per sustain, portamento e via dicendo), in cui c è posto a zero (off) oppure a $7F (on). Tali unità di controllo corrispondono ai normali joystick per computer dotati di interruttori. I numeri di controllore da $60 a $79 non sono definiti e quelli da $7A a $7F sono riservati per i messaggi di modo dei canali. Non esiste alcuna restrizione per l'abbinamento dei numeri di controllore a specifici dispositivi fisici, comunque è consuetudine associare alla ruota di modulazione il numero di controllore $1.
5. Il controllo locale è usato, eventualmente, per interrompere il collegamento interno tra la normale unità d'immissione di uno strumento (ad esempio la tastiera) e i circuiti generatori di suono, cosicché la tastiera, per esempio, invia dati soltanto all'uscita MIDI OUT e il circuito generatore di suoni reagisce soltanto ai segnali applicati alla MIDI IN.
6. L'implementazione della condizione "tutte le note off" è opzionale e si consiglia di non adoperare tali messaggi al posto dei normali comandi per l'annullamento di singole note. Tuttavia i fabbricanti non sembrano avere idee chiare in merito a questi messaggi, rendendone l'uso virtualmente inutile.
7. Il terzo byte del messaggio per il modo mono specifica il numero totale di canali richiesto. Se questo numero è zero, il numero di canali eguaglia il numero di voci disponibili sul ricevitore.
8. La "ruota" usata per modificare l'altezza delle note è un dispositivo di controllo diverso da tutti gli altri, poiché può produrre valori sia positivi sia negativi. La posizione centrale equivale a l = $0 e m = $40. Tuttavia, molti ricevitori hanno soltanto 7 bit di risoluzione, pertanto reagiscono soltanto alle variazioni di m, ignorando quelle di l.

MESSAGGI DI SISTEMA

Stato	Dato1	Dato2	Descrizione	Note
$F0	qualsiasi	numero	Esclusivo di sistema	1
$F1			Non definito	
$F2	l	m	Puntatore alla posizione della melodia l = 7 Bit Meno Significativi m = 7 Bit Più Significativi	2
$F3	s		Selettore di melodia s = numero della melodia	
$F4			Non definito	
$F5			Non definito	
$F6			Richiesta di accordatura	3
$F7			Fine di esclusivo di sistema	1
$F8	—	—	Impulsi di sincronizzazione	4
$F9			Non definito	
$FA	—	—	Inizio	4
$FB	—	—	Continuazione	4
$FC	—	—	Arresto	4
$FD			Non definito	
$FE	—	—	Individuazione di attività	5
$FF	—	—	Reset del sistema	6

Note:

I messaggi da $F8 a $FF sono quelli "in tempo reale" e possono essere inviati in qualsiasi momento (anche durante l'esecuzione di altri messaggi).

1. I messaggi esclusivi di sistema, che possono contenere un qualsiasi numero di byte di dati, terminano con l'apposito codice di "fine messaggio esclusivo" ($F7) oppure con un qualsiasi altro byte di stato.
2. Questo messaggio è adoperato per predisporre arbitrariamente il puntatore al brano da eseguire. Questo puntatore è un registro interno, che contiene il numero di battute (1 battuta = 6 impulsi di clock MIDI) a partire dall'inizio della sequenza (brano da eseguire).
3. Questo messaggio serve per richiedere l'accordatura degli oscillatori interni ai sintetizzatori analogici.
4. Questi messaggi vengono usati per sincronizzare l'unità sequenzatrice principale e quelle secondarie. Il temporizzatore di sistema viene regolato su una durata pari a 1/24 di un quarto di nota. Il comando "continuazione" differisce dal comando "inizio" in quanto l'esecuzione riprende dal valore corrente del puntatore al brano, anziché ripartire dal principio.
5. Il messaggio di "individuazione di attività" è adoperato, all'occorrenza, come comando "di comodo", quando il sistema MIDI non svolge alcuna attività. Se adoperato, occorre inviarlo in modo che non trascorrano più di 300ms senza lo svolgersi di una qualche attività. Il ricevitore dovrebbe comportarsi normalmente se non riceve alcun segnale di "individuazione di attività", ma se lo riceve e trascorrono più di 300ms senza attività, esso dovrebbe spegnere tutte le voci per poi tornare allo stato normale.
6. Questo messaggio è adoperato per inizializzare l'intero sistema subito dopo l'accensione. L'inizializzazione non può essere eseguita in modo automatico all'accensione, altrimenti c'è il rischio che due o più componenti del sistema continuino ad inizializzarsi a vicenda all'infinito.

I messaggi che non compaiono nell'elenco qui riportato non devono essere trasmessi, tanto più che non dovrebbero (almeno in teoria) essere riconosciuti dai ricevitori. In particolare, occorre accertarsi che non vengano emessi messaggi spuri al momento dell'accensione.

L'Apricot F1e

Anatomia di un microcomputer a diffusione europea: tecnologie a confronto

L'aspetto del modello F1e è piuttosto familiare a chi già conosce i computer della serie Apricot. La sua configurazione di base comprende la tastiera F1, l'unità centrale (che incorpora l'unità a dischetti a singola faccia e singola densità) e un monitor a fosfori verdi da 8 pollici. Come nell'Apricot Portable (vedere a pag. 1589), anche nel modello F1e la connessione tra tastiera e unità centrale è ottenuta tramite raggi infrarossi, eliminando così la necessità di adoperare appositi cavetti. Le informazioni trasmesse sia dalla tastiera sia dal "mouse" (opzionale) vengono captati da un sensore collocato sul pannello anteriore dell'unità centrale. Seguendo i più recenti dettami estetici (di cui il Macintosh della Apple è stato il precursore), l'Apricot F1e ha un ingombro ridotto al minimo, così da poter essere comodamente appoggiato su una scrivania, senza occupare troppo spazio. L'unità centrale è di forma rettangolare con una profondità maggiore rispetto alla larghezza. La configurazione della tastiera non si discosta dai normali standard della Apricot: i tasti non sporgono dal corpo della tastiera e ciò la fa assomigliare alla tastiera del QL. Il difetto comune a queste due tastiere è la scarsa comodità d'impiego da parte di esperti dattilografi o di chi intenda cimentarsi in lughe sessioni di word-processing. Inoltre, poiché i tasti tendono ad avere un certo gioco nelle loro sedi, può sorgere qualche dubbio sull'affidabilità di tutto l'insieme.

Nella parte superiore della tastiera sono presenti, come nell'Apricot Portable, quattro tasti speciali: il RESET (che serve per la "partenza a freddo" del sistema), il REPEAT RATE (da adoperare per variare la velocità di ripetizione dei tasti quando questi vengono tenuti premuti), il SET TIME (per regolare l'ora) e il KEYBOARD LOCK.

Il computer dispone di una sola unità a dischetti da 3 1/2' formato Sony, situata nella parte anteriore dell'unità centrale. L'uso di questo formato va diffondendosi sempre più e, poiché la ACT è stata una delle prime aziende ad adottarlo, non deve adesso affrontare il grosso problema comune a molti produttori di computer: il trasferimento del software esistente dal vecchio al nuovo formato; gli acquirenti di F1, pertanto, non si trovano a corto di programmi. Inoltre, sebbene a singola faccia e singola densità, i dischetti usati hanno una capacità di ben 315 Kbyte, superiore alla maggior parte dei dischetti da 5 1/4 pollici.



CHRIS STEVENS

Sul pannello frontale dell'unità centrale è collocata una serie di LED che indicano se l'apparecchio è acceso, se sono attive le opzioni di Caps Lock e di scroll e, infine, se l'unità a dischetti è in funzione. Sotto i LED si trovano i sensori a raggi infrarossi.

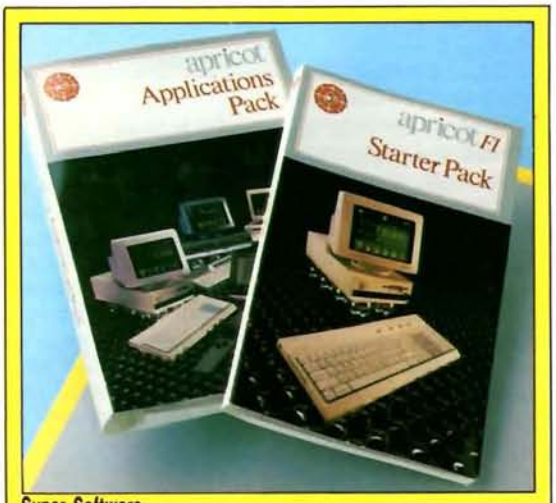
Alla destra dell'unità centrale è situato un bus di espansione a 60 vie, tramite il quale è possibile collegare una vasta gamma di schede aggiuntive, oltre a una seconda unità a dischetti. Per di più, utilizzando il sistema di espansione MSD, il computer può contare su di disco rigido da 10 Megabyte portandosi così al pari del modello Apricot XI.

Il pannello posteriore, benché non offra molte interfacce, è progettato in modo da consentire il collegamento delle più comuni periferiche. All'estrema sinistra c'è un connettore standard tipo D a 25 vie per l'interfaccia seriale RS232. Accanto a questa porta si trovano due prese per monitor. La prima, a

Estetica da futuro

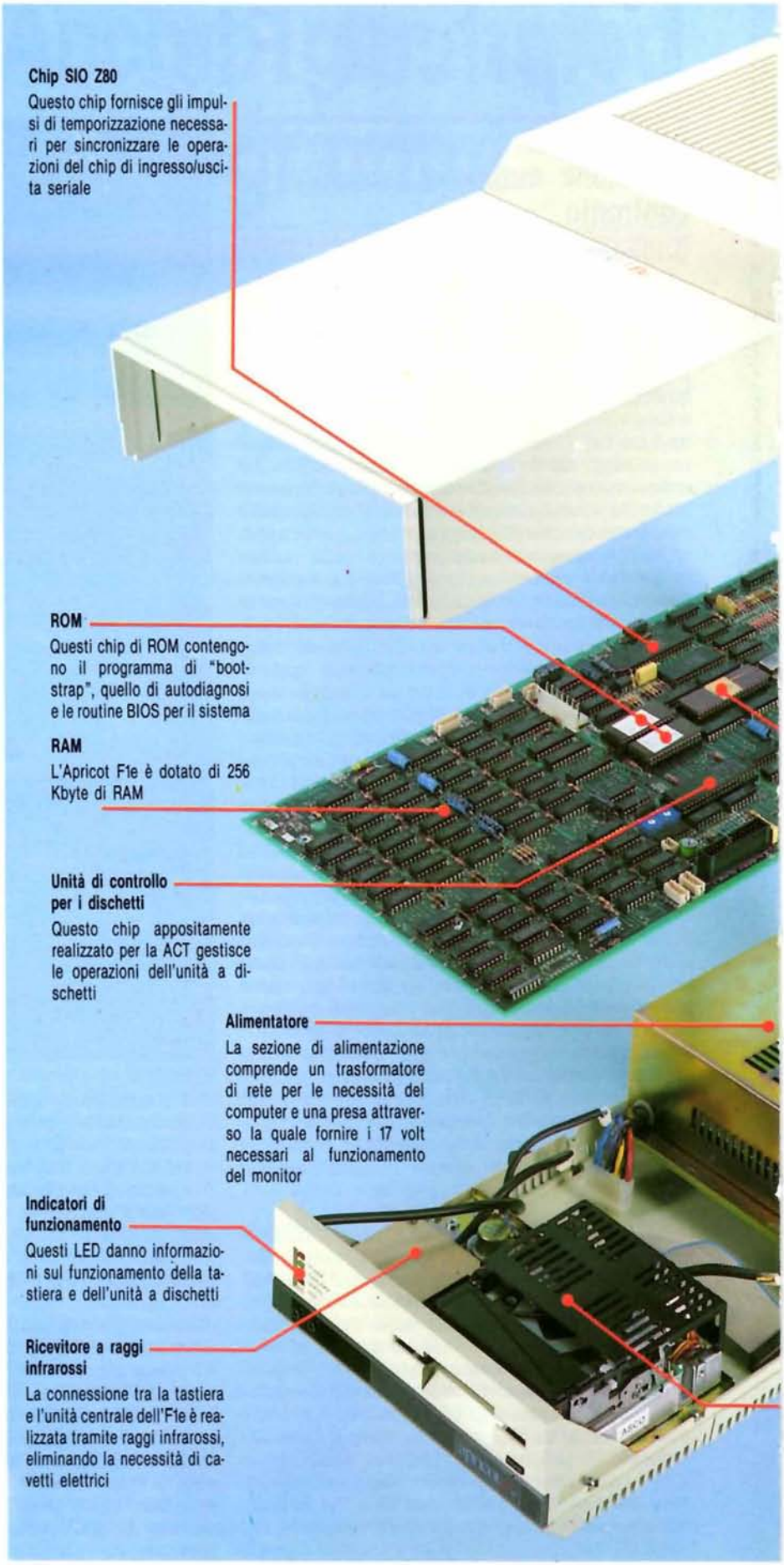
L'Apricot F1e ha ereditato l'ottimo design che caratterizza anche i suoi predecessori. L'unità per dischetti da 3 1/2 pollici, il sistema operativo MS-DOS e il software fornito di corredo (oltre ad un prezzo molto ragionevole) fanno di questo computer un apparecchio che si adatta molto bene sia ad applicazioni commerciali sia al settore della didattica

nove piedini, fornisce i segnali per monitor a colori RGB (i monitor monocromi della ACT, però vengono collegati a questa stessa presa). La seconda è un'uscita per video composito, adatta ad altri tipi di monitor.
Il monitor è venduto a parte e, sebbene non esista un'uscita RF adatta ai comuni televisori, è possibile acquistare un apposito adattatore. In ogni caso, le uscite per monitor non prevedono una linea di alimentazione, né l'unità centrale possiede prese a tal scopo. Ciò significa che anche acquistando un monitor ACT occorre procurarsi un alimentatore esterno da 17 volt.
Ciò contrasta con l'impostazione avveniristica di tutto il resto del progetto (con particolare riferimento all'impiego di raggi infrarossi al posto dei tradizionali cavetti). Oltretutto, altri modelli della stessa casa non presentano lo stesso inconveniente ed è una magra consolazione il fatto che tutto sommato l'alimentatore, che può essere nascosto alla vista, non turbi l'estetica globale dell'apparecchio.
Internamente al computer esiste un ulteriore connettore per schede aggiuntive; la scheda principale del computer è ben progettata e, come tutte quelle della ACT, è protetta da eventuali interferenze da parte del trasformatore di alimentazione e dal motore delle unità a disco. Tale protezione è ottenuta tramite una custodia metallica che funge anche da dissipatore di calore.
La CPU impiegata nell'F1e è l'ormai ben noto chip a 16 bit 8086, col quale è possibile far girare il sistema operativo MS-DOS. Tuttavia, tutti i modelli Apricot hanno sempre posseduto un proprio sistema di gestione (chiamato *Activity*) basato su icone per cui i contatti diretti con il sistema operativo dell'apparecchio sono comunque ridotti al minimo.
Il software *Activity* è un cosiddetto "program-



Super Software

Il gruppo di programmi forniti di corredo all'Apricot F1e comprende: un word-processor, una specie di agenda elettronica e uno spreadsheet. Mentre i primi due (il "SuperWriter" e il "SuperPlanner") non offrono prestazioni al di sopra dello standard, il programma "SuperCalc" possiede caratteristiche molto sofisticate, che ne fanno un prodotto di livello altamente professionale



Chip SIO Z80
Questo chip fornisce gli impulsi di temporizzazione necessari per sincronizzare le operazioni del chip di ingresso/uscita seriale

ROM
Questi chip di ROM contengono il programma di "bootstrap", quello di autodiagnosi e le routine BIOS per il sistema

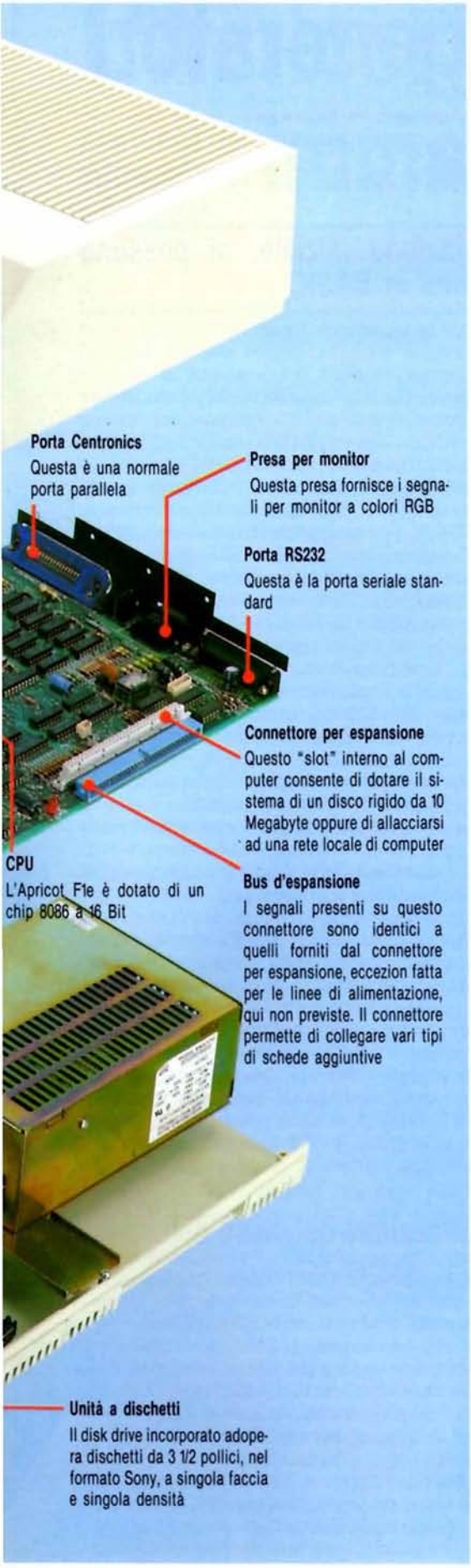
RAM
L'Apricot F1e è dotato di 256 Kbyte di RAM

Unità di controllo per i dischetti
Questo chip appositamente realizzato per la ACT gestisce le operazioni dell'unità a dischetti

Alimentatore
La sezione di alimentazione comprende un trasformatore di rete per le necessità del computer e una presa attraverso la quale fornire i 17 volt necessari al funzionamento del monitor

Indicatori di funzionamento
Questi LED danno informazioni sul funzionamento della tastiera e dell'unità a dischetti

Ricevitore a raggi infrarossi
La connessione tra la tastiera e l'unità centrale dell'F1e è realizzata tramite raggi infrarossi, eliminando la necessità di cavetti elettrici



CHRIS STEVENS

ma orientato all'oggetto". Ciò significa che, anziché dover impartire tutta una serie di comandi (quali ad esempio quelli necessari per eseguire la LOAD di un programma in memoria), basta indicare, generalmente posizionando il cursore su di una tra le icone visualizzate sullo schermo, quale azione si desidera intraprendere. Il computer pensa a tutto il resto.

In genere i sistemi operativi basati su icone implicano necessariamente l'impiego di un mouse, ma sull'Apricot è stata prevista una soluzione alternativa: la tastierina numerica; poiché i tasti sono disposti su tre file di tre tasti ciascuna, ogni tasto (escluso quello centrale) corrisponde ad una delle otto direzioni possibili. Posizionato il cursore sull'icona desiderata, l'esecuzione viene avviata premendo il tasto ENTER, a fianco della tastierina numerica.

Chi ritenesse di trovarsi in difficoltà nell'uso di un sistema operativo con simili caratteristiche può avvalersi di un programma appositamente progettato per familiarizzare con l'*Activity*. Esso spiega, fra l'altro, come utilizzare i set di caratteri alternativi esistenti e come crearne di nuovi, e inoltre come configurare il sistema per l'impiego delle periferiche ad esso collegate.

Il software di corredo

Come è ormai consuetudine in apparecchi destinati ad applicazioni commerciali, anche il modello F1e è corredato da una notevole quantità di programmi. Il *SuperWriter* è un word-processor piuttosto sofisticato, anche se non raggiunge le vette del notissimo *WordStar*. Il programma *SuperPlanner* è descritto come un blocco per appunti elettronico, che consente di pianificare i propri impegni. Comprende anche un programma per la gestione di un indirizzario, un calendario, un'agenda e un piccolo sistema di archiviazione dati. Sebbene offra qualche caratteristica tipica dei database, il *SuperPlanner* non consente criteri di ricerca e di estrazione particolarmente sofisticati.

Il *SuperCalc*, infine, è l'"immancabile" lavagna elettronica per applicazioni finanziarie e contabili. Questo è forse il package più completo tra quelli in dotazione: contiene numerosi comandi per l'apertura di "finestre", per il centramento dei testi visualizzati e tanti altri ancora. Il *SuperCalc* può essere definito un programma "spreadsheet" altamente professionale.

Sebbene questo computer sia particolarmente adatto ad applicazioni commerciali, la ACT non intende evidentemente sottovalutare il mercato della didattica nel proprio paese di origine, la Gran Bretagna. Infatti, assieme all'annuncio di un calo dei prezzi di listino, è stato lanciato un prodotto, il B-TRAN, che consente di far girare sull'F1e, senza alcuna modifica, i programmi scritti nel BASIC del BBC, per cui esiste una quantità di software vastissima.

APRICOT F1e

DIMENSIONI

mm. 425 x 200 x 105

CPU

Intel 8086, funzionante a 4,7 MHz

MEMORIA

256 Kbyte di RAM come standard

VIDEO

Modo testuale: 132 x 50 oppure 80 x 25 caratteri.
Massima risoluzione nel modo grafico: 800 x 400 pixel

INTERFACCE

Porta parallela Centronics, seriale RS232. Uscite per monitor RGB e video composito

UNITÀ A DISCHETTI

Singola unità a 3 1/2 pollici, formato Sony a singola faccia e singola densità con capacità di 315 Kbyte

SISTEMI OPERATIVI

MS-DOS, CP/M-86 e CP/M Concurrent

DOCUMENTAZIONE

Il manuale relativo ai programmi applicativi è compilato con precisione ed è ricco di informazioni utili, mentre quello per i principianti è un po' troppo sbrigativo e superficiale

PRO

Poiché il prezzo dell'apparecchio è altamente concorrenziale, questo computer può essere adatto sia al settore commerciale sia a quello didattico

CONTRO

La tastiera "piatta" non è certo tra le migliori per prolungate sessioni di word-processing. Il funzionamento del computer è leggermente più lento rispetto ad altre macchine che adoperano la medesima CPU



Ancora generatori di programmi

Con un'attenta progettazione iniziale, si possono produrre ottimi programmi in BASIC

Il "fai da te" dei programmi

"Sycero" e "The Last One" ("TLO") consentono la generazione di programmi BASIC "trasportabili", ossia eseguibili anche su apparecchi diversi da quello sul quale sono stati creati. In ogni caso, però, la struttura e le specifiche del programma da generare vanno definite con molta attenzione durante le fasi preliminari, se si vogliono ottenere risultati apprezzabili

Ogni generatore di programmi per i più inesperti deve possedere requisiti fondamentali:

- Facilitare l'applicazione di una progettazione di tipo "top-down", durante la quale l'utente traccia a grandi linee il flusso del programma desiderato.
- Possedere un'impostazione basata sull'uso di menu, dai quali l'utente possa scegliere le opzioni necessarie allo sviluppo.
- Prevedere un sistema di immediata individuazione e correzione degli errori e permettere piccole modifiche al progetto.
- Il codice risultante dovrebbe essere "trasportabile", ossia utilizzabile anche su altri computer indipendentemente dal software che lo ha generato.
- Fornire un'ampia documentazione per permettere la comprensione del processo di generazione del programma e, se necessario, la modifica o il miglioramento del prodotto finale.

Tuttavia i generatori di programmi, adoperati anche da specialisti, non dovrebbero essere appesantiti troppo da macchinose procedure di verifica e di controllo, altrimenti il tempo per ottenere risultati apprezzabili non è più molto vantaggioso rispetto ai metodi di programmazione tradizionali. Infine, il generatore deve consentire l'inclusione di particolari subroutine fornite dall'operatore, che spesso i più esperti preferiscono a quelle prodotte dai generatori.

I due package in commercio che meglio si adattano a tutti i requisiti fin qui esposti sono *Sycero* e *The Last One* (*TLO*).

In entrambi vengono profusamente impiegati menu molto descrittivi. *Sycero* può visualizzare diverse pagine di informazioni concernenti, ad esempio, il controllo del cursore; i comandi di edit sono costantemente disponibili: basta premere i tasti Control e H. Dei due sistemi, il *TLO* è quello che segue più fedelmente il criterio "top-down". Il lavoro inizia con la stesura di una specie di diagramma di flusso del programma da generare. In sostanza questo si riduce a una serie di frasi REM, che spiegano la funzione di ciascuna parte del programma e che possono essere incorporate nel listato finale.

Gli schermi per l'immissione e la visualizzazione di dati vengono definiti durante il processo di codifica e il codice risultante è in formato ASCII. Per eseguire il programma generato, l'utente deve uscire dal generatore, eseguire una LOAD del programma risultante e quindi una SAVE in formato binario. Col *Sycero*, il primo compito è la definizione dei file e degli schermi usati dal programma da generare. Questa fase è seguita dall'abbinamento di messaggi alle varie operazioni di input previste e da diverse altre definizioni. I moduli vengono poi collegati assieme prima della codifica.

Sebbene un simile modo di procedere sia poco agevole per i meno esperti, garantisce che la codifica del programma avvenga soltanto se le fasi preliminari sono state svolte correttamente e integralmente: individuato qualche errore durante la codifica, il generatore si ferma ed informa subito l'utente, che può quindi provvedere senza perder tempo. Finita la codifica, il programma viene automaticamente memorizzato in formato binario e può essere eseguito dall'interno del *Sycero*.

Il codice prodotto è trasportabile; se il programma finale è breve, la versione creata dal *TLO* (in formato ASCII) può essere eseguita anche su apparecchi MSX. Entrambi i generatori, però, includono nel programma finale routine di verifica e controllo aumentandone le dimensioni.

Identificazione del disco di lavoro

Come tutti i programmi di buona qualità, sia il *Sycero* che il *TLO* sono accompagnati da uno speciale programma di installazione, che adatta il software alle caratteristiche dell'hardware impiegato (ad esempio la presenza di due unità a dischetti oppure di un disco rigido). Con il *TLO*, inoltre, si deve "identificare" il disco di lavoro. Questa procedura somiglia in parte al formattamento di un nuovo dischetto. Il termine "disco di lavoro" può riferirsi sia a un intero floppy disk, sia a una porzione (o "subdirectory") di un disco rigido. Questa identificazione iniziale è di fondamen-

tale importanza, poiché altrimenti il package può non ritrovare, durante la fase di correzione, il diagramma di flusso relativo al progetto in corso. In tal caso l'utente non ha grandi alternative: o ricomincia tutto da capo o tenta di intervenire sul listato BASIC prodotto dal generatore. A scanso di equivoci, comunque, conviene sempre eseguire una copia integrale del dischetto di lavoro.
Sia il *TLO* che il *Sycero* producono un'ampia documentazione, contenente informazioni quali le caratteristiche degli schermi usati, i nomi delle variabili adoperate e via dicendo. Il *Sycero*, inoltre, aggiunge la data e l'ora sia sullo schermo sia sui listati finali, consentendo così di riconoscere facilmente la più recente tra più versioni di uno stesso programma.
Eseguita l'installazione, all'utente di *TLO* viene presentato un menu principale che comprende otto opzioni:

Creazione di programma	Ricerca
Modifica di programma	Identificazione nuovo dischetto
Modifica di file	Ripresa della codifica
Definizione di file	Ritorno al BASIC

Dopo aver selezionato la prima opzione e aver risposto affermativamente alla domanda «Il programma prevede l'uso di file?», occorre definire le caratteristiche (lunghezza e tipo dei dati per ciascun campo) dei file previsti. Terminata questa fase, il programma presenta il menu che consente di redigere il diagramma di flusso. Nella più recente versione di *TLO* questo menu offre 20 opzioni:

Lista del flow-chart	Annullamento
Modifica al flow-chart	Definizione dei puntatori ai file
Codifica del programma	Lettura di un file
Fusione di flow-chart	Scrittura di un file
Fine lavoro	Ricerca o riordino di un file
Immissione da tastiera	Fusione
Visione di dati	Verifica dei record
Diramazioni	Azzeramento file
Calcoli	Funzioni per database
Funzioni speciali	Funzioni multiple

Usando queste opzioni si può creare un diagramma di flusso, ad esempio per un indirizzario, che può somigliare al seguente:



La codifica può iniziare. Fin qui, il generatore non consente di uscire senza perdere il lavoro fatto. Appena inizia la codifica, però, è possibile interrompere il lavoro per poi riprenderlo usando l'apposita opzione per la Ripresa della codifica. Durante la fase di codifica, si stabiliscono gli indirizzi delle diramazioni. Il primo menu, ad esempio, dirama al punto 2 (Scrittura dei dati), al 7 (Lettura dei dati) oppure al 18 (Conclusione). Inoltre vengono costruiti gli schermi di immissione e di visualizzazione.
È possibile (e consigliabile) eseguire una SAVE degli schermi, così da riutilizzarli in seguito altrove, nello stesso o in altri programmi. Il manuale del *TLO*, tuttavia, non pone sufficiente enfasi su questo punto, mentre nel *Sycero* l'operazione è automatica.

Il menu d'apertura di "Sycero"

Il menu iniziale di *Sycero* offre 13 opzioni, usate più o meno nell'ordine di presentazione.

Configurazione del sistema
Inizializzazione
Definizione dei file-campi di sistema
Definizione degli schermi
Elaborazione degli schermi
Definizione dei rapporti
Elaborazione dei rapporti
Definizione di un programma
Generazione di un programma
Creazione di un file di dati "dal vivo"
Esecuzione di un programma generato
Utilità
Fine della sessione

Quando si raggiunge l'opzione per la generazione del programma, gran parte del lavoro è già stato svolto: ciò che segue non richiede grossi interventi, salvo nel caso di individuazione di errori. Poiché, nessun generatore possiede ancora doti di chiaroveggenza, quindi ciò che si desidera ottenere dal programma finale va definito con molta precisione.
La principale differenza tra i due generatori considerati è nell'approccio alla progettazione. Come osserva il manuale del *Sycero*, il miglior modo per sviluppare un sistema è quello di applicare un criterio "top-down": definire per prime le linee generali del programma e quindi passare alla definizione dei particolari. Sfortunatamente, però, la maniera migliore per immettere tali dettagli in un generatore di programmi è proprio seguendo il percorso inverso, ossia il "bottom-up".
Poiché inizia dalla fase di stesura del flow-chart, il *TLO* è considerato un esempio migliore di progettazione "top-down". Tuttavia, l'utente può essere tentato di fornire soltanto un abbozzo sbrigativo, per passare subito ai dettagli. *Sycero*, al contrario, non produce risultati codificati finché non è terminata correttamente ogni fase preliminare, ammettendo anche "ripensamenti" durante tali fasi.
È auspicabile che, in un futuro prossimo, questo settore vada ampliandosi, magari usufruendo dei risultati prodotti dagli studi sull'Intelligenza Artificiale.

I supercomputer

Un confronto tra l'incredibile Cray-1 e un sistema con processore Z80



Un computer da sogno

Il Cray-1 (illustrato qui sopra) offre una potenza elaborativa inimmaginabile, sotto il controllo di un computer "ospitante", quale un IBM, un DEC e simili. L'architettura interna del Cray-1 (vedere diagramma in basso) è caratterizzata da 32 Megabyte di memoria principale e da 4.888 byte di spazio riservato ai registri della CPU. Il tutto è realizzato con l'impiego di 3.400 schede di circuito, collegate tra loro per mezzo di ben 90 km. di cavi

La "potenza" di un computer dipende da diversi fattori: la lunghezza della "parola" usata nel processore, la velocità di trasferimento dei dati, le dimensioni della memoria centrale e il ciclo di lavoro della CPU del sistema. Nei microcomputer, le funzioni principali della CPU sono tutte racchiuse in un unico microprocessore, ad esempio: il notissimo Z80 della Zilog, gli 8088 e 8086 della Intel e l'M68000 della Motorola. Tutti ricorrono alla tecnologia MOS (semiconduttori a ossido di metallo) per i circuiti logici e la memoria incorporata nel chip. I dati vengono trasferiti ed elaborati in parallelo a 8 o 16 bit alla volta. Le frequenze di lavoro dei microprocessori variano da 1MHz a 12 MHz.

Tuttavia le caratteristiche elencate, fanno dei computer basati su microprocessori apparecchi non adatti all'elaborazione di enormi quantità di dati previste da applicazioni quali il trattamento di immagini, la dinamica dei fluidi o le previsioni meteorologiche.

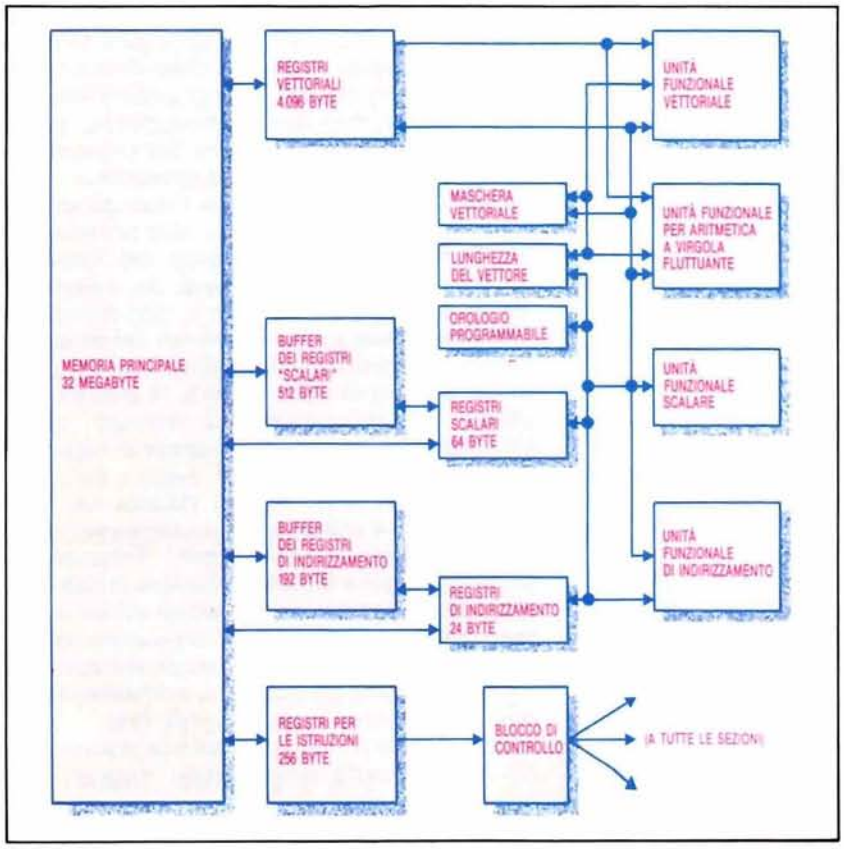
Ecco un esempio che rende bene l'idea. Si supponga di voler realizzare un film adoperando un computer per creare grafica d'animazione, in cui ogni "quadro" debba avere una risoluzione di 6000 x 6000 pixel e siano richiesti 24 "quadri" al secondo. Poiché si tratta di grafica d'animazione, ogni punto di un'immagine può cambiare posizione tra un quadro e l'altro, cosicché la posizione di ciascun punto va calcolata di volta in volta. Ciò implica l'esecuzione di circa 864 milioni di calcoli al secondo. Per di più, ogni espressione calcolata può essere complessa e richiedere quindi dozzine o centinaia di istruzioni in codice macchina. Vale a dire qualcosa come qualche miliardo di operazioni al secondo. Nel riquadro della pagina a fianco è descritto un confronto approssimativo tra il tempo per produrre una sequenza animata di dieci minuti sia su di un micro basato sullo Z80 sia su di un supercomputer.

I supercomputer (così vengono chiamati i più grossi mainframe) lavorano secondo criteri abbastanza simili a quelli dei microcomputer: le istruzioni e i dati vengono estratti dalla memoria centrale, i dati sono elaborati dal processore in base alle istruzioni impartite e, infine, i risultati vengono depositati in memoria. La differenza principale è individuabile nella quantità dei dati elaborabili e nella velocità di elaborazione, oltre a diversità nell'architettura interna dei due tipi di computer.

Il Cray-1 S/4400, della Cray Research, è un esempio tipico di supercomputer. Il primo Cray-1 è stato installato nel 1976, a soli quattro anni dalla formazione dell'azienda, e da allora la fama di questi apparecchi è andata diffondendosi nel mondo. La CPU del Cray-1 occupa un intero armadietto cilindrico, alto quasi due metri, realizzato in modo da sembrare un divanetto circolare (nei "sedili" sono alloggiati i sistemi di raffreddamento e di alimentazione). La sua enorme velocità deriva dall'impiego della "logica a semiconduttori bipolari" (sono "semplici" transistor, a differenza dei semiconduttori MOS, CMOS, NMOS, FET e via dicendo). La logica bipolare e la memoria del Cray sono racchiuse in oltre 200.000 circuiti integrati montati su 3.400 schede. I relativi collegamenti hanno richiesto oltre 90 km. di filo elettrico.

La lunghezza della "parola" elaborabile dalla CPU è di 64 bit (otto volte quella del microprocessore Z80) e il clock di sistema viaggia a 80 MHz (80 milioni di cicli al secondo). Per eseguire una somma a 64 bit bastano 37,5 nanosecondi. Una somma a 8 bit, su di uno Z80 funzionante a 4 MHz, richiede 1,75 microsecondi. La memoria principale, nel Cray, fa parte della CPU e contiene 32 Megabyte, organizzati in 4.194.304 parole. I trasferimenti in memoria avvengono ad un ritmo di 2.560 milioni di byte al secondo.

Come è logico attendersi, la CPU del Cray dispone di un enorme numero di registri



interni: i registri di indirizzamento (tutti a 24 bit) sono 72, i registri scalari (tutti a 64 bit) sono anch'essi 72 e i registri vettoriali (ognuno lungo 64 parole), sono 8. Lo spazio totale occupato dai registri della CPU del Cray è pertanto di 4.888 byte, contro i 26 dello Z80.

L'interfaccia con l'utente

Diversamente da molti computer, che operano come unità a sé stanti, i Cray sono progettati per funzionare come estensioni di altri minicomputer o mainframe. Questi agiscono da "interfaccia" tra utente e potenza elaborativa dei Cray, occupandosi dell'immissione dai terminali o da lettori di schede e dell'emissione su stampanti o su unità di registrazione magnetica. Tra la CPU del Cray e il computer che funge da interfaccia è situato il "sottosistema di I/O" del Cray, progettato per conformarsi agli standard delle varie marche di computer (IBM, DEC, Data General o altri). Il sottosistema di I/O impiega due o più processori dedicati alle operazioni di ingresso/uscita, ciascuno dei quali può essere considerato come un vero e proprio microcomputer estremamente potente.
Questa breve descrizione del Cray chiarisce appena la sua architettura interna (il diagramma a blocchi della CPU può far intuire quanto il sistema sia sofisticato). Del resto, un'esposizione delle istruzioni disponibili, delle 13 "unità funzionali" capaci di lavorare in parallelo o del criterio di elaborazione vettoriale, che consente di intervenire simultaneamente su ben 64 coppie di operandi, oltre a richiedere tanto spazio, interessa soprattutto quei pochi fortunati che hanno possibilità di acquistare quello che spesso è stato definito il computer più potente del mondo.

APPLICAZIONI DEI SUPERCOMPUTER

Previsioni meteorologiche

Le attuali previsioni meteorologiche sono il frutto di un'elaborazione di dati raccolti in tutte le parti del mondo, anche tramite satelliti, e di adeguati modelli computerizzati. La realizzazione di un "modello" di portata mondiale implica l'esecuzione di miliardi di operazioni di calcolo su vaste matrici di dati e i risultati devono essere disponibili nell'arco di poche ore. Soltanto computer immensamente potenti quanto il Cray-1 sono in grado di gestire simili situazioni

Dinamica dei fluidi

La dinamica dei fluidi riveste un ruolo primario nel progetto di automobili in cui l'utilizzazione del carburante deve esser ottimale, di sistemi di raffreddamento per centrali nucleari, di realizzazioni aeronautiche, per citare solo alcuni settori. L'analisi e la previsione del comportamento dei fluidi richiede l'elaborazione di vastissime quantità di dati. Se poi i risultati devono essere prodotti in tempo reale, le cose si complicano ancora di più. Il problema è che ogni particella di fluido (e ve ne possono essere innumerevoli miliardi) ha una propria influenza su tutte le altre particelle del fluido in un sistema in movimento. La potenza elaborativa richiesta ha dimensioni colossali

Previsioni economiche

È noto quanto sia difficile la creazione di modelli economici validi. Così come accade nella dinamica dei fluidi, anche in questo caso l'effetto globale deriva dal comportamento di ogni singola particella del sistema. Il tentativo di realizzare un modello a livello mondiale ha scarse applicazioni pratiche, ma, anche limitandosi a modelli più ristretti, la complessità è di vaste proporzioni. Anche in questo caso, è necessario l'impiego di computer dotati di enormi capacità elaborative, se si vogliono ottenere risultati utilizzabili a fini pratici e non semplicemente "storici"

TEMPI A CONFRONTO

Per paragonare la potenza elaborativa di un supercomputer Cray-1 con quella di un home computer basato sullo Z80 (quale l'Amstrad CPC 464), si consideri la realizzazione di un film di 10 minuti, utilizzando una grafica d'animazione ad alta risoluzione (di 6000 x 6000 pixel) e 24 "quadri" al secondo.
Si supponga che, per ogni quadro, debba essere calcolata la posizione di ogni suo punto e che tale calcolo, sullo Z80, implichi l'esecuzione di 100 istruzioni in codice macchina, ognuna delle quali richieda mediamente 19 cicli di lavoro (4,75 microsecondi). Per il Cray-1, che dispone di istruzioni molto più sofisticate, si immagini che ogni calcolo implichi 25 istruzioni in codice macchina (esagerando per eccesso), ognuna delle quali richieda un tempo medio di esecuzione di 4 cicli di lavoro (50 nanosecondi).
Con uno Z80 il tempo occorrente sarebbe: $6000^2 \times 24 \times 60 \times 10 \times 100 \times 4{,}75 \times 10^{-6} = 2{,}4624 \times 10^8$ secondi, ossia 7,8 anni. Con un Cray-1 le cose sono ben più accettabili: $6000^2 \times 24 \times 60 \times 10 \times 25 \times 50 \times 10^{-9} = 6{,}48 \times 10^5$ secondi, ossia 7,5 giorni!



Per concessione di MIKE MANSFIELD ENTERPRISES

Rivoluzione visuale

La grafica generata su computer sta invadendo tutti i settori, in particolare quelli della produzione pubblicitaria, cinematografica e televisiva. Immagini come quelle qui raffigurate vengono realizzate molto facilmente usando i più recenti minicomputer. Per generare oltre 20 minuti di scenario per un film di fantascienza, però, è stato necessario ricorrere alla potenza del Cray-1 (lo stesso risultato, ma su di un micro ad 8 bit, avrebbe richiesto ben 15 anni di lavorazione!)



Microdrive e porta seriale

Altre utili routine in codice macchina disponibili per il programmatore

L'accesso ai Microdrive in codice macchina può avvenire sia tramite i codici "di aggancio" (vedere a fianco) sia eseguendo chiamate dirette alle routine ROM dell'Interface 1. Tuttavia, questa seconda tecnica è sconsigliabile, poiché al sistema può essere collegata più di una ROM per l'Interface 1. Pertanto, di seguito vengono esaminati i 13 codici "di aggancio" relativi all'uso dei Microdrive.

Prima di far ciò è bene ricordare alcuni principi sul funzionamento delle unità Microdrive. Dopo il formattamento, una cartuccia risulta suddivisa in 255 "settori", che si possono considerare come l'unità di misura "fisica" per l'archiviazione dei dati. Ogni settore è ulteriormente suddiviso in una "testata" (che contiene, tra l'altro, il nome assegnato alla cartuccia); un blocco di dati (contenente, oltre ad altre informazioni, il nome del file che occupa il settore); un gruppo di altri dati e, infine, un record di dati lungo 512 byte. La procedura di formattamento provvede, inoltre, a marcare adeguatamente eventuali settori danneggiati o inutilizzabili.

Come tutte le altre forme di ingresso/uscita sullo Spectrum, anche quelle riguardanti i Microdrive avvengono tramite un canale che, nel caso specifico, è costituito da un'area di memoria lunga 595 byte. Abbinata a tale zona ce n'è un'altra, chiamata "mappa" del Microdrive, composta da 32 byte. Questa viene creata dal sistema operativo per riconoscere quali settori della cartuccia sono occupati (oppure inutilizzabili) e quali sono invece disponibili per la scrittura di nuovi dati.

La zona di memoria riservata al canale per il Microdrive appartiene alla cosiddetta "Area dei Canali per Microdrive". Ogni volta che viene predisposto un canale, la memoria compresa tra la fine dell'Area dei Canali per Microdrive e l'indirizzo STKEND è spostata verso l'alto, liberando uno spazio sufficiente. Quando il canale non serve più esso viene chiuso e la porzione di memoria torna ad essere disponibile per altri usi.

I canali per Microdrive vengono predisposti in due modi: "esplicitamente" quando un file viene aperto con un comando OPEN oppure "implicitamente" quando il sistema operativo si accinge ad eseguire un'operazione sul canale del Microdrive (ad esempio, quando si adopera il comando SAVE). Un canale aperto esplicitamente deve essere in seguito chiuso altrettanto esplicitamente, mentre nell'altro caso esso viene chiuso automaticamente dal sistema così come è stato aperto.

La struttura di un canale per Microdrive è riassunta nel diagramma riportato nella pagina a fianco. Si noti la somiglianza dei primi byte con quelli per i canali dello schermo e della tastiera, esaminati in precedenza (vedere a pag. 1797). La maggior parte delle informazioni relative al canale per Microdrive, comunque, interessa solo in modo marginale, poiché è raro dovervi accedere direttamente. L'indirizzo della mappa del Microdrive, dalla quale si possono estrarre utili informazioni sulla distribuzione e sull'occupazione dei dati sulla cartuccia, è contenuto nel canale. Del resto, è proprio così che opera il comando CAT del BASIC. Volendo esaminare la mappa di un Microdrive, si può adoperare il seguente programma (dopo aver impartito una NEW):

```
 10 OPEN #5;"M";1;"prova"
 20 inizio = PEEK(23870) + 256*PEEK(23871)
 30 FOR I = inizio TO inizio + 31
 40 LET settore = PEEK(I)
 50 FOR J = 1 TO 8
 60 IF settore/2 = INT(settore/2) THEN PRINT "0";
 70 IF settore/2 <> INT(settore/2) THEN PRINT "1";
 80 LET settore = settore/2 : LET settore = INT(settore)
 90 NEXT J
100 PRINT
110 NEXT I
120 CLOSE #5
```

Il programma visualizza 1 per indicare un settore occupato e 0 per segnalare un settore libero. Con poche modifiche questo programma può essere usato per visualizzare la quantità di spazio libera sulla cartuccia.

L'area di 512 byte riservata ai dati è utilizzata dal sistema operativo per il trasferimento, appunto, dei dati. I dati vengono scritti sulla cartuccia quando il relativo buffer è pieno, quando viene impartito un comando CLOSE# oppure quando viene richiamato il codice "di aggancio" relativo a tale comando. In quest'ultimo caso, il contenuto del buffer viene memorizzato sul nastro, immediatamente seguito dal codice di "fine file" (EOF). In seguito viene esaminato un ulteriore caso relativo alla scrittura del buffer dei dati. In condizioni normali, dunque, i dati vengono scritti sul nastro della cartuccia in blocchi di 512 byte.

Vediamo adesso come controllare le operazioni su Microdrive.

Codici di aggancio per Microdrive

Codice di aggancio 33: questo serve per avviare il motore del Microdrive specificato, il cui numero identificativo è compreso tra 1 e 8. L'operazione è chiamata anche "selezione del drive". Questo codice può essere adoperato anche per spegnere il motore a tutti i

Microdrive collegati. Si noti che la routine può disabilitare le interrupt, quindi è buona norma riabilitarle sempre, con l'struzione EI, dopo una chiamata a questo codice.

Per usare la routine basta caricare nel registro A il numero del Microdrive interessato ed usare una RST #8. Caricando in A il valore 0 vengono spenti tutti i motori dei Microdrive. In alternativa, i motori possono essere spenti generando un errore in BASIC. Questa routine non funziona in due casi: se allo Spectrum non è collegato alcun Microdrive oppure se si tenta di avviare il motore di un Microdrive che non contiene una cartuccia formattata.

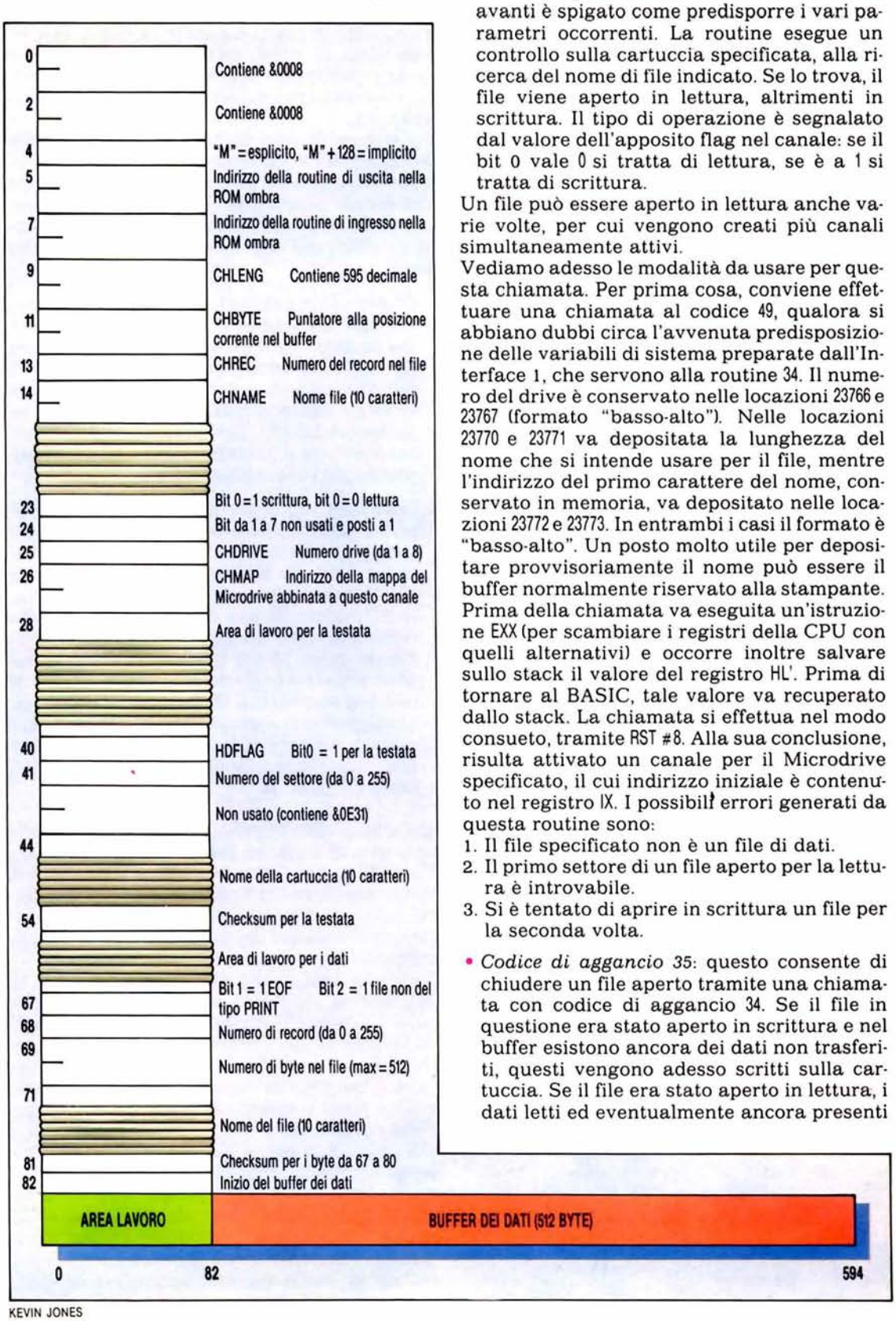
- *Codice di aggancio 34*: questa chiamata consente di aprire un file sulla cartuccia. Più avanti è spigato come predisporre i vari parametri occorrenti. La routine esegue un controllo sulla cartuccia specificata, alla ricerca del nome di file indicato. Se lo trova, il file viene aperto in lettura, altrimenti in scrittura. Il tipo di operazione è segnalato dal valore dell'apposito flag nel canale: se il bit 0 vale 0 si tratta di lettura, se è a 1 si tratta di scrittura.

Un file può essere aperto in lettura anche varie volte, per cui vengono creati più canali simultaneamente attivi.

Vediamo adesso le modalità da usare per questa chiamata. Per prima cosa, conviene effettuare una chiamata al codice 49, qualora si abbiano dubbi circa l'avvenuta predisposizione delle variabili di sistema preparate dall'Interface 1, che servono alla routine 34. Il numero del drive è conservato nelle locazioni 23766 e 23767 (formato "basso-alto"). Nelle locazioni 23770 e 23771 va depositata la lunghezza del nome che si intende usare per il file, mentre l'indirizzo del primo carattere del nome, conservato in memoria, va depositato nelle locazioni 23772 e 23773. In entrambi i casi il formato è "basso-alto". Un posto molto utile per depositare provvisoriamente il nome può essere il buffer normalmente riservato alla stampante. Prima della chiamata va eseguita un'istruzione EXX (per scambiare i registri della CPU con quelli alternativi) e occorre inoltre salvare sullo stack il valore del registro HL'. Prima di tornare al BASIC, tale valore va recuperato dallo stack. La chiamata si effettua nel modo consueto, tramite RST #8. Alla sua conclusione, risulta attivato un canale per il Microdrive specificato, il cui indirizzo iniziale è contenuto nel registro IX. I possibili errori generati da questa routine sono:

1. Il file specificato non è un file di dati.
2. Il primo settore di un file aperto per la lettura è introvabile.
3. Si è tentato di aprire in scrittura un file per la seconda volta.

- *Codice di aggancio 35*: questo consente di chiudere un file aperto tramite una chiamata con codice di aggancio 34. Se il file in questione era stato aperto in scrittura e nel buffer esistono ancora dei dati non trasferiti, questi vengono adesso scritti sulla cartuccia. Se il file era stato aperto in lettura, i dati letti ed eventualmente ancora presenti



KEVIN JONES

Struttura del canale per Microdrive

Un canale per Microdrive occupa 595 byte della memoria. Il suo indirizzo iniziale è contenuto nel registro IX a seguito della chiamata con codice di aggancio = 34

nel buffer vengono perduti. A seguito della chiamata, infatti, viene liberata la porzione di memoria occupata dal canale. La routine va chiamata previo caricamento nel registro IX dell'indirizzo iniziale del canale interessato. Anche in questo caso, è buona norma salvare il contenuto del registro HL' prima della chiamata. L'unico errore previsto può presentarsi se sulla cartuccia non c'è spazio sufficiente per chiudere il file.

- *Codice di aggancio 36*: questo codice consente di cancellare un file dalla cartuccia, sia esso composto da dati o da un programma. Il nome del file, il suo indirizzo e la sua lunghezza, assieme al numero di drive, sono passati con lo stesso criterio adottato per la chiamata 34.

Spiegato come sia possibile aprire, chiudere e cancellare i file, rimane da vedere come effettuarne la lettura e la scrittura. Ovviamente si può abbinare un flusso al canale del Microdrive, ma è molto più comodo far sì che quest'ultimo diventi il canale "corrente" di lettura/scrittura (sostituendo, rispettivamente, i normali flussi provenienti dalla tastiera o diretti allo schermo).
Pertanto la maniera più semplice per eseguire una scrittura consiste nell'usare una RST #10, dopo aver fatto diventare il canale del Microdrive quello corrente di scrittura depositando l'indirizzo iniziale del canale del Microdrive nella variabile di sistema CURCHL, in 23633 e 23634. Il comando occorrente è:

LD (23633),IX

dove IX contiene l'indirizzo iniziale del canale, fornito dalla chiamata 34. L'esecuzione della RST #10, adesso, provoca il dirottamento di ogni successiva scrittura verso il canale del Microdrive, anziché verso lo schermo. La semplicità di questo procedimento sottolinea la troppo spesso sottovalutata elasticità del SO dello Spectrum. Come accennato in precedenza, l'effettiva scrittura sulla cartuccia avviene quando il buffer del canale si riempie. Per scrivere in un file chiamato PIPPO, ad esempio, il codice necessario è il seguente:

```
        ; scrittura del file "PIPPO" su Microdrive
3E01            ld   a,1            ;numero del drive
32D65D          ld   (23766),a      ;depositato in memoria
AF              xor  a              ;azzera il registro A
32D75C          ld   (23767),a
212B29          ld   hl,nome        ;hl punta nome file
22DC5C          ld   (23772),hl     ;ne deposita l'indirizzo
210400          ld   hl,4           ;4 = lunghezza del nome
22DA5C          ld   (23770),hl     ;deposita la lunghezza
D9              exx                 ;scambia i registri CPU
E5              push hl             ;salva hl' sullo stack
D9              exx
CF              rst  #8
22              defb 34             ;codice di aggancio 34
DD22515C        ld   (23633),ix     ;CURCHL = canale del drive
06FF            ld   b,255          ;255 byte nel file
C5      ciclo:  push bc             ;salva il contatore
3E00            ld   a,CHAR         ;qui va inserita la routine
                                    ;che estrae i byte da scrivere
                                    ;dalla zona di memoria in cui
                                    ;li ha depositati l'utente
D7              rst  #10            ;scrittura nel canale
C1              pop  bc             ;recupera il contatore
10F6            djnz ciclo
CF              rst  #8
23              defb 35             ;chiude il file
3E02            ld   a,2
CD0116          call #1601          ;emissione su schermo
D9              exx
E1              pop  hl             ;recupera HL
D9              exx
FB              ei   ;riabilita le interrupt
C9              ret
46524544 nome:  defm "PIPPO"        ;nome del file
```

Ovviamente, se si apre più di un file l'indirizzo iniziale di ciascun canale va depositato in una zona sicura di memoria, affinché possa essere riutilizzato più volte. La routine appena descritta crea sulla cartuccia un file lungo 255 byte.
La lettura di dati da tale file non presenta problemi: il file viene aperto in modo identico e in CURCHL viene depositato l'indirizzo iniziale del canale Microdrive. Per la lettura di ogni carattere si usa la routine di sistema all'indirizzo #15E6, che deposita nel registro A il byte letto.

- *Codice di aggancio 37*: questa chiamata, previo caricamento in IX dell'indirizzo iniziale del canale Microdrive, esegue una lettura del record successivo nel file, depositandolo nell'apposita area del canale. Se la lettura avviene regolarmente, il valore di CHREC viene incrementato. Ovviamente, il nuovo record letto va a sovrapporsi ai dati eventualmente già presenti nel canale.

- *Chiamata di aggancio 38*: questo codice provvede alla scrittura su cartuccia dei dati contenuti nel canale Microdrive, in modo simile a quanto avviene per la lettura. Come al solito, IX deve contenere l'indirizzo iniziale del canale. Si noti che se un settore della cartuccia viene riempito di dati solo parzialmente (meno di 512 byte), il SO dello Spectrum considera tale settore libero, se non si usa l'accorgimento di terminare i dati aggiungendovi il codice di fine file (EOF). Ciò si ottiene adoperando, anziché la chiamata con codice 38, quella per la chiusura del file, ossia il codice 35.

Un'ultima utile routine di lettura è abbinata al codice 39: legge un particolare record all'interno di un file. Come di consueto il registro IX deve contenere l'indirizzo iniziale del canale, mentre la variabile di sistema CHREC deve contenere il numero del record che si desidera leggere.
Esistono altri codici di aggancio relativi ai Microdrive, ma il loro uso è rarissimo. Il codice 40, ad esempio, esegue la lettura di un intero settore di una cartuccia, il cui numero va depositato in CHREC. Il codice 41 serve per leggere il "successivo" settore della cartuccia e il codice 42 per scrivere nel primo settore libero della cartuccia.
Gli ultimi due codici qui esaminati riguardano la manipolazione dei canali per Microdrive. Il codice 43 dovrebbe produrre un canale e una mappa per la cartuccia. Tuttavia, nella prima versione delle ROM per l'Interface 1, questa routine svolgeva la stessa funzione del codice 34. Nelle versioni successive la chia-

mata funziona regolarmente. Il codice 44 rimuove un canale (il cui indirizzo si trova in IX) e la mappa ad esso abbinata.
Per concludere l'analisi dei Microdrive, un ultimo avvertimento: gli esperimenti con i codici di aggancio vanno eseguiti con cartucce che non contengono dati importanti, poiché può capitare per sbaglio di danneggiare il formattamento della cartuccia, nel qual caso... addio dati!

Le routine per la porta seriale RS232

Il connettore dell'interfaccia seriale RS232 è dotato di nove piedini (nel diagramma alla pagina seguente è indicata la funzione di ciascuno di essi). Il numero di linee da utilizzare varia a seconda del tipo di applicazione. Il collegamento che prevede il minor numero di connessioni si avvale delle linee 2, 3 e 7. In tal caso, le linee 4 e 5 andrebbero collegate al piedino 9, in modo da mantenerle a livello logico alto, altrimenti può accadere che lo Spectrum si fermi in attesa del segnale indicante che il dispositivo collegato alla porta è pronto a ricevere dati. Lo svantaggio di un collegamento ridotto al minimo come questo è che possono andar persi dati se, durante la trasmissione, il dispositivo ricevente è momentaneamente impreparato alla ricezione.
Una possibile applicazione della porta RS232 è quella di dotare lo Spectrum di una stampante "vera". Effettuati i collegamenti necessari, il software di controllo è molto semplice da realizzare. Per inviare testi alla stampante, occorre selezionare la corretta velocità in baud (tramite il comando FORMAT) e poi usare il seguente codice:

```
CLOSE #3
OPEN #3;"T"
```

Ciò abbina il flusso 3, normalmente riservato alla stampante ZX, al canale T (quello per il testo sulla porta RS232). Una dettagliata descrizione sulla distinzione tra flussi testuali e binari è fornita nel manuale dell'Interface 1. A questo punto, i comandi LLIST e LPRINT producono il loro effetto sulla stampante collegata all'Interface 1.
Passiamo adesso ai dettagli sul funzionamento dell'interfaccia seriale. I dati inviati hanno il seguente formato: un bit di start, otto bit di dati e due bit di stop. Comunque, non occorre preoccuparsi del formato, perché a questo pensa il sistema operativo. È invece importante concentrare l'attenzione sul valore dei baud. La velocità di trasmissione viene generalmente modificata tramite il comando FORMAT, che prevede un certo numero di valori standard. Benché l'interfaccia RS232 perda i dati abbastanza difficilmente anche su distanze ragguardevoli, nel caso di collegamenti particolarmente lunghi è sempre meglio usare velocità minori.
Prima di intervenire con le tecniche descritte qui di seguito è bene accertarsi che le variabili di sistema occorrenti vengano predisposte opportunamente. Ciò si ottiene adoperando il codice di aggancio 49. I codici relativi all'interfaccia RS232 sono due, mentre le variabili di sistema interessate sono tre. Esaminiamole per prime.

- BAUD (locazioni 23747 e 23478). Questa è una variabile a 16 bit il cui valore determina la velocità di trasferimento dei dati. Qui si incontra una limitazione dello Spectrum: non si possono avere due velocità diverse (una per la trasmissione e un'altra per la ricezione).
- SERFL (locazione 23751). Questo è un flag usato dal SO soltanto durante la ricezione. Se posto a 1, significa che è ancora disponibile un byte per la lettura.
- SERBYT (locazione 23752). Questo byte è usato come buffer sia in ingresso che in uscita. Può darsi che in questa locazione venga depositato un byte dopo che si è interrotta la comunicazione.

Gli inconvenienti cui sono soggette queste ultime due variabili vengono esaminati in seguito.
Una quarta variabile di sistema, ossia IOBORD (alla locazione 23750), riguarda la comunicazione seriale solo in modo marginale: essa specifica il colore del bordo dello schermo durante le operazioni di trasferimento. Il colore può essere scelto a piacimento.
Volendo usare la porta RS232, dunque, la prima operazione da eseguire è la scelta della

VELOCITÀ IN BAUD E INTERFACE 1

Lo standard RS232 è diffusamente impiegato nelle comunicazioni tra computer, anche di marche diverse, in particolare quando la distanza tra gli apparecchi è grande. Il numero di baud indica la velocità alla quale avvengono i trasferimenti. Applicando la formula riportata nel testo, si può ottenere qualsiasi velocità si desideri. Ecco, comunque, le velocità più usate assieme ai valori ottenuti dalla formula:

Velocità Baud	Valore
50	2690
110	1221
300	446
600	222
1200	110
2400	54
4800	28
9600	12
19200	5

Il valore va depositato, con una POKE, nella variabile di sistema BAUD, situata in 23747 e 23748. La presenza di un'unica variabile di sistema, purtroppo, impedisce la selezione di due diverse velocità (una per la trasmissione e una per la ricezione in comunicazioni simultanee nei due sensi).
Il seguente brano di codice illustra come selezionare una velocità baud (nel caso specifico 300 baud):

```
LD    HL,446
LD    (23747),HL
RET
```

velocità in baud: basta depositare il valore desiderato nella variable BAUD (ed è ciò che si limita a fare il comando FORMAT). Il valore viene così calcolato:

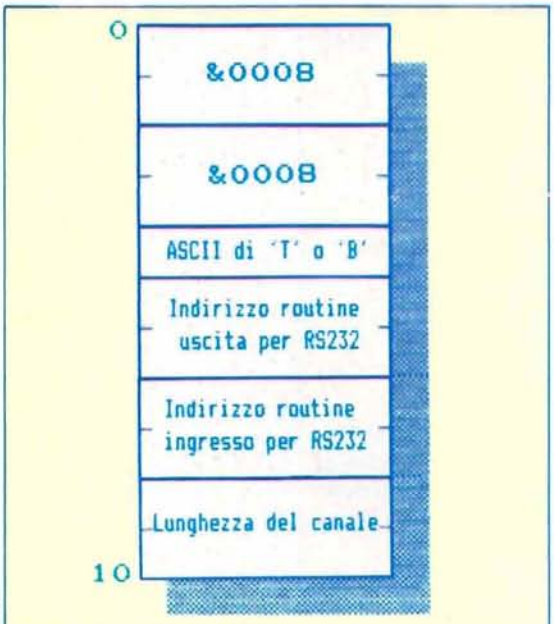
```
valore = (3500000/(26*velocità in baud))-2
```

Usando questa equazione, è possibile selezionare velocità diverse da quelle standard, anche se ciò è di scarsa utilità se il secondo apparecchio collegato non ha una simile possibilità. Le velocità più comunemente usate, e i relativi valori, sono riportati nella tabella alla pagina precedente?.

Vediamo adesso come avviene l'invio dei dati. In BASIC, l'apertura (OPEN) di un flusso diretto alla RS232 provoca la preparazione di un apposito canale, nell'Area dei canali, il cui formato è riportato nella tavola? qui sotto. Al valore del byte 4 di questo canale viene aggiunto 128 se il canale è stato creato implicitamente (ad esempio per mezzo dei comandi SAVE*, MOVE, FORMAT, ecc.).

Si noti che non esiste, nel canale, un'area riservata al buffer dei dati. I dati vengono scritti o letti chiamando l'appropriato codice di aggancio che, a sua volta, richiama la routine il cui indirizzo si trova nella quinta e sesta locazione (per la trasmissione) o nella settima e ottava locazione (per la ricezione) del canale. Usando i codici di aggancio il canale non viene predisposto automaticamente. I due codici per la porta seriale RS232 sono:

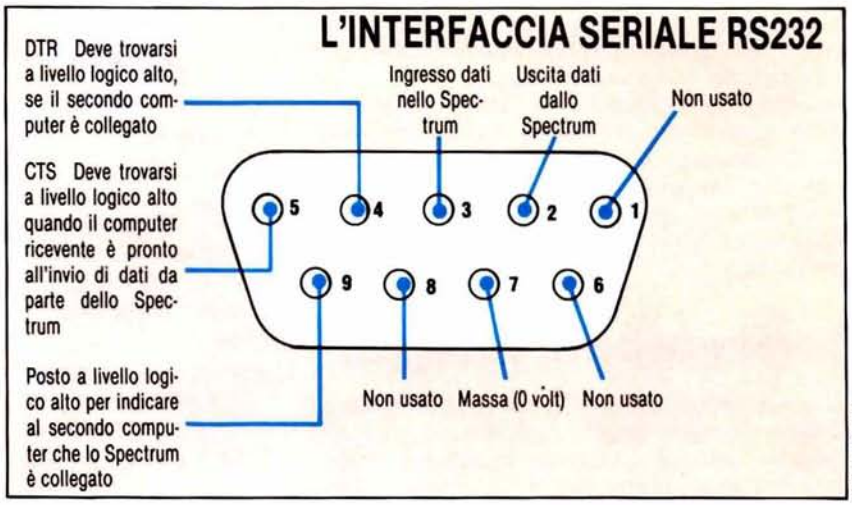
- *Codice di aggancio 29*: questa chiamata deposita nel registro A il carattere letto dalla porta seriale, purché ve ne sia uno disponibile (fatto segnalato dal flag di carry posto a 1). Se non ci sono caratteri disponibili, sia il registro A che il flag di carry valgono 0.



CAROLINE CLAYTON

Struttura del canale per la RS232

Si noti l'assenza di un buffer per l'appoggio dei dati da trasferire. Tutti i dati da inviare o da ricevere vengono passati mediante il registro A della CPU, usando i codici di aggancio 29 e 30



KEVIN JONES

- *Codice di aggancio 30*: questa chiamata trasmette alla porta seriale il byte contenuto nel registro A, usando la velocità selezionata in precedenza. A questo livello, non esiste alcuna distinzione tra i flussi di testo e quelli binari.

È a questo punto che le due variabili SERBYT e SERFL possono causare inconvenienti. Se la routine abbinata al codice di aggancio trova il valore 1 in SERFL, essa estrae qualsiasi valore si trovi in SERBYT (anche se questo è un residuo di una precedente comunicazione). Questo fatto può anche compromettere la validità di tutti i byte che vengono letti in seguito, pertanto è buona norma azzerare SERFL, prima di effettuare una qualsiasi lettura dalla porta seriale. Ecco il codice occorrente:

```
XOR A
LD (23751),A
```

Questo è l'unico vero inconveniente che si può incontrare nell'uso dell'interfaccia seriale. Nel caso non si riesca ad ottenere nessun tipo di comunicazione, si provi a scambiare tra loro i cavetti collegati ai piedini 2 e 3 dell'Interface 1. La velocità in baud, sebbene non sia estremamente stabile, rientra nella tolleranza ammessa dallo standard RS232.

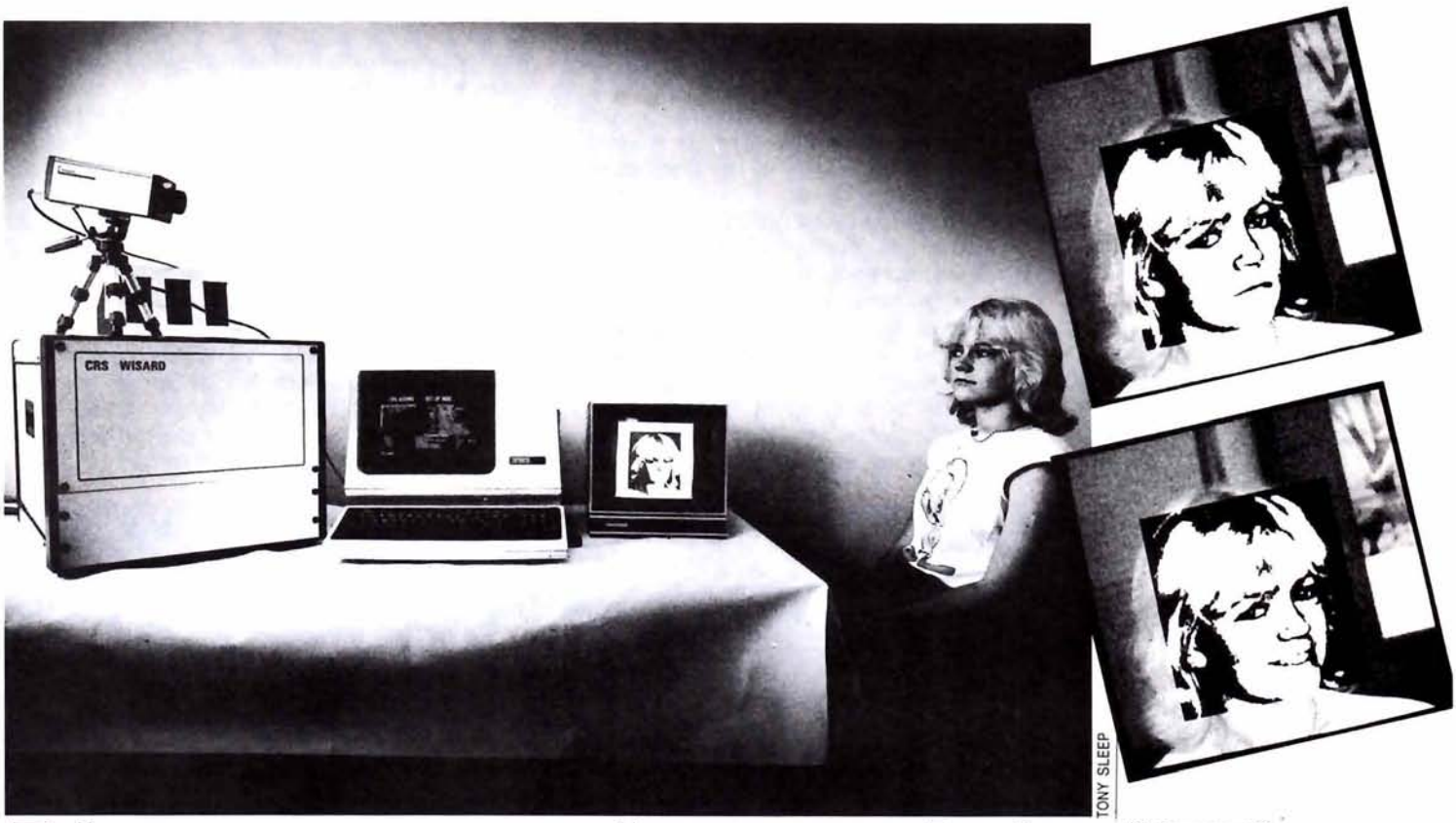
Ecco un esempio di subroutine per l'invio di 255 byte alla porta seriale RS232:

```
         ;invio di 255 byte verso la porta seriale RS232
         ;
CF       rst  #8          ;preparazione delle
31       defb 49          ;variabili di sistema
216E00   ld   hl,110      ;valore per 1200 Baud
22C35C   ld   (23747),hl  ;fissa la velocità in baud
06FF     ld   b,255       ;prepara il contatore
ciclo:   push bc          ;salva il contatore
3E00     ld   a,DATO      ;qui va inserita la routine
CF       rst  #8          ;che estrae i byte da inviare
1E       defb 30          ;invio del byte in A
C1       pop  bc          ;recupero del contatore
10F8     djnz ciclo
C9       ret
```

Per molti versi, la connessione tramite la porta RS232 è la funzione più semplice da ottenere dall'Interface 1: non occorre preoccuparsi della preparazione né della gestione di canali ed esistono soltanto due chiamate (una per la trasmissione e una per la ricezione).

Mezzi di comunicazione

La porta seriale dell'Interface 1 consente allo Spectrum di comunicare con altri computer o dispositivi seriali, mediante un connettore standard a nove piedini di tipo D. L'unico vero inconveniente della porta RS232 dello Spectrum è l'impossibilità di fissare diverse velocità di trasmissione e di ricezione simultaneamente



Di che umore sei?
Il sistema "Wisard", sviluppato da Igor Aleksander presso l'Imperial College di Londra, usa una griglia di 512 x 512 pixel e un Megabyte di RAM per riconoscere le immagini visualizzate da un televisore. Grazie alla sua elevata sensibilità, il "Wisard" è in grado di distinguere anche le diverse espressioni di un volto

Riconoscere immagini

La "vista" del computer e un programma dimostrativo in BASIC

Nei sistemi per il riconoscimento di immagini che prevedono la tecnica "bottom-up", dalla sagoma originale vengono estratte alcune caratteristiche significative con lo scopo di ridurla ad una figura più semplice e di significato più generico. Uno dei metodi usati in questa tecnica implica i cosiddetti "operatori locali" per effettuare la scansione dell'immagine. Ciascun operatore esamina una porzione della figura e moltiplica i valori delle tonalità grigie per un "fattore di peso". Il prodotto assume valori alti quando il contorno sembra appartenere ad una sagoma conosciuta dal sistema. Adoperando più operatori locali, si riesce ad individuare ed estrarre caratteristiche come linee orizzontali, verticali e oblique, ma anche intere zone d'ombra e di luce.
Il programma *Wisard* creato da Igor Aleksander (vedere a pag. 1862), applica all'immagine osservata il principio della "generazione di indirizzo di memoria", che consente di memorizzarla e, in seguito, riconoscerla.
Il principale vantaggio offerto dal *Wisard* è l'indirizzamento "in parallelo" di tutti i banchi RAM, che consente di elaborare ad alta velocità immagini televisive fortemente contrastate. Il programma dimostrativo in BASIC (a pagina seguente) simula un sistema di questo tipo e, pur operando molto più lentamente, ne evidenzia i principi di funzionamento.
Il sistema di riconoscimento usa discriminatori a griglie di scansione "sestuple" (che sono meno efficaci delle griglie "ottuple" ma permettono di risparmiare memoria). Le 40 sestuple impiegate richiedono 20.480 bit di RAM, ossia 2.560 byte, in quanto ciascuna può trovarsi in 64 stati diversi e indirizzare un banco RAM di 64 locazioni. Ogni locazione contiene 8 bit che consentono al sistema di distinguere otto diverse classi d'immagini.
Il programma si svolge in due fasi: nella prima, detta di "addestramento", il computer riceve una serie di esempi relativi alle varie classi, mentre nella seconda esegue il riconoscimento dell'immagine confrontandola con gli esempi memorizzati.
Supponiamo che, durante la fase di addestramento, la sestupla numero 20 generi il valore 110010 (50 in decimale) in risposta a un'immagine di classe 4. In questo caso, il sistema seleziona il quarto bit della locazione 50 del banco RAM numero 20. Se, durante il riconoscimento, il discriminatore genera nuovamente il decimale 50, trova il valore 1 nel quarto bit dell'indirizzo chiamato, il che conferma la presenza dell'immagine di classe 4.

Il sistema immagazzina i dati dell'immagine

in una matrice di byte (azzerata dalla subroutine 1000 all'inizio di ogni elaborazione), che inizia dall'indirizzo D. La subroutine alla linea 2000 permette di creare semplici immagini 16×16, usando i tasti A, B, S e D per muovere il cursore e il carattere * e la barra spaziatrice per comporre la figura.
Il programma può essere modificato per caricare e salvare la matrice D su disco o su nastro ma, nella versione corrente, è necessario ripetere la procedura di classificazione delle immagini ogni volta che il programma viene eseguito. In versione applicativa, il sistema interagisce col mondo esterno tramite un convertitore analogico/digitale collegato a una telecamera (un sistema di questo tipo è descritto a pag. 1229).

Si noti che l'assegnazione dei pixel ai discriminatori sestupli avviene in modo casuale e deve essere ripetibile.
L'informazione dell'immagine viene inizialmente registrata in forma binaria nella matrice bidimensionale I(,): i valori 0 e 1 corrispondono rispettivamente allo spazio e all'asterisco (*). Durante la fase di addestramento gli indirizzi vengono generati da elementi di I(,) scelti in modo casuale, mentre il bit corrispondente viene selezionato alla linea 210 in base alla classe trattata in quel momento. Durante il processo di riconoscimento, il programma deve generare l'identica sequenza di indirizzi casuali per rendere possibile il confronto tra l'immagine in esame e il sistema di classificazione memorizzato.

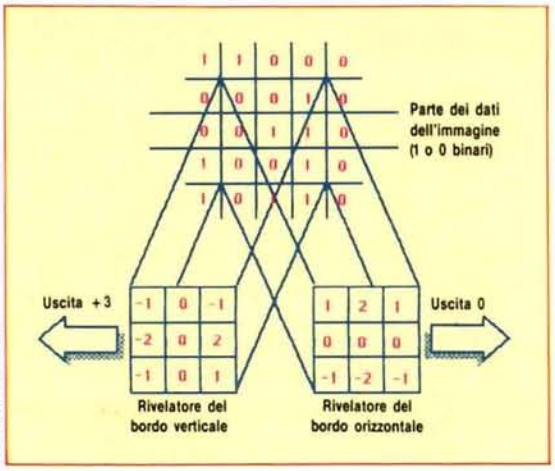
PROGRAMMA PER RICONOSCERE LE IMMAGINI

Micro BBC:

```
10 REM **********************************
30 REM ** RICONOSCIMENTO DELL'IMMAGINE **
40 REM ** CON PROCEDURA WISARD PER BBC **
50 REM **********************************
55 MODE 7
56 SX%=40: DS%=6: RB%=2^DS%
57 MEMO%=SX%*RB%
60 DIM I%(16,16), CS%(7)
64 DIM D% MEMO%
65 @%=4: REM formato o/p
66 REM D% e' una matrice di byte
70 REM I% e' la matrice dell'immagine, D% quella degli
   indirizzi
80 REM Si noti che I% contiene 0 o 1 binari
85 PRINT "Dimensioni della RAM = "; MEMO%; " byte."
88 GOSUB 1000: REM azzera la matrice D%
90 REM -- Fase di Addestramento:
92 MOOD$="Addestramento"
95 INPUT "Tipo di classe (0-7)? ",C%
96 IF C%<0 OR C%>7 THEN GOTO 90
100 FOR I%=1 TO 16: FOR J%=1 TO 16: I%(I%,J%)=0
101 NEXT: NEXT
110 REM -- Accetta l'immagine tracciata dall'utente:
120 GOSUB 2000
130 R=RND(-1): REM valore casuale
140 FOR I%=1 TO SX%
150 A%=RB% * (I%-1)
155 REM A% e' l'indirizzo di base del banco RAM
160 FOR J%=0 TO (DS%-1)
170 R1%=INT(RND(1)*16+1)
180 R2%=INT(RND(1)*16+1)
190 A%=A%+I%(R1%,R2%)*2^J%
200 NEXT J%
202 REM R1% e R2% sono coordinate 'casuali'
210 PROCdset(A%,C%)
220 NEXT J%
230 INPUT "Un'altra fase di addestramento (S/N)? ",A$:
    IF A$="S" THEN GOTO 90
233 IF A$="s" THEN GOTO 90
235 MOOD$="Riconoscimento"
236 REM -- Fase di riconoscimento:
240 FOR I%=1 TO 16: FOR J%=1 TO 16
244 I%(I%,J%)=0
245 NEXT: NEXT
250 REM -- Accetta un'immagine da classificare:
260 GOSUB 2000
270 R=RND(-1)
280 FOR C%=0 TO 7: CS%(C%)=0: NEXT
290 FOR I%=1 TO SX%
300 A%=RB% * (I%-1)
310 FOR J%=0 TO DS%-1
320 R1%=INT(RND(1)*16+1)
330 R2%=INT(RND(1)*16+1)
340 A%=A%+I%(R1%,R2%) * 2^J%
350 NEXT
355 FOR C%=0 TO 7
360 IF FNdget(A%,C%) > 0 THEN CS%(C%)=CS%(C%)+1
370 NEXT C%
380 NEXT I%
382 CX%=0
385 FOR C%=0 TO 7
388 PRINT "La Classe ";C%;" ha un valore di ";CS%(C%)/
    SX%*100;SPC(8)
390 IF CS%(C%)>CS%(CX%) THEN CX%=C%
400 NEXT C%
404 PRINT "La classe piu' somigliante e' la n.";CX%
410 INPUT "Vuoi classificare un'altra immagine (S=Si)?
    ";A$
420 IF A$="S" OR A$="s" THEN GOTO 240
440 PRINT: PRINT "Fine!"
444 END
999 :
1000 REM -- Inizializzazione della memoria:
1010 FOR I%=0 TO MEMO%
1020 REM -- Usa byte come indice (?):
1030 ?(D%+I%)=0
1040 NEXT
1050 RETURN
1199 :
1200 DEF PROCdset(A%,C%)
1210 REM -------------
1220 ?(D%+A%) = (D%?A%) OR 2^C%
1230 ENDPROC
1240 REM Seleziona un bit della matrice D%
1244 :
1250 DEF FNdget(A%,C%)
1260 REM -------------
1270 =(D%/A%) AND  2^C%
1280 REM Estrae un bit dalla matrice D%
1288 REM A% e' l'indirizzo; C% e' compreso tra 0 e 7
1290 :
2000 REM -- Routine di creazione dell'immagine
2010 CLS: PRINT TAB(0,20);"Usare: A,B,S,D,X,* o la
     barra spaziatrice"
2012 PRINT "per comporre l'immagine.)"
2013 PRINT "Fase di ",MOOD$
2015 PRINT TAB(1,1);
2020 T%=0: A$=" "
2030 H%=1: V%=1
2040 REPEAT C$=INKEY$(222)
2050 IF C$ = "A" THEN V% = V% - 1
2060 IF C$ = "B" THEN V% = V% + 1
2070 IF C$ = "S" THEN H% = H% - 1
2080 IF C$ = "D" THEN H% = H% + 1
2090 IF H%>16 THEN H%=1
2095 IF H%<1 THEN H%=16
2100 IF V%>16 THEN V%=1
2105 IF V%<1 THEN V%=16
2110 IF C$=" " THEN T%=0: A$=C$
2120 IF C$="*" THEN T%=1: A$=C$
2130 I%(H%,V%)=T%
2140 PRINT TAB(H%,V%); A$;
2150 UNTIL C$="X"
2155 GOSUB 2200: REM Visualizza l'immagine creata
2160 RETURN
2170 :
2200 REM -- Routine di visualizzazione:
2210 FOR H%=1 TO 16
2220 FOR V%=1 TO 16
2230 IF I%(H%,V%)>0 THEN PRINT TAB(H%,V%);":" ELSE
     PRINT TAB (H%,V%);" ";
2240 NEXT: NEXT
2250 PRINT TAB(0,17);
2260 RETURN
2270 :
2300 REM Funzione dei tasti:
2301 REM A & B per cursore ALTO/BASSO
2302 REM D & S per cursore DES/SIN
2303 REM * & Barra spaz. per ACCESO/SPENTO
2304 REM X per uscire (dopo aver creato l'immagine).
```

Commodore 64:

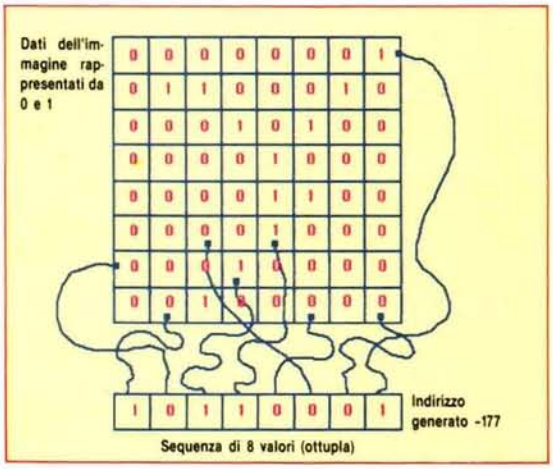
```
10 REM ** RICONOSCIMENTO DELL'IMMAGINE **
20 REM **        PER COMMODORE 64       **
30 FOR I=1 TO 25: DW$=DW$+CHR$(17): NEXT I
56 SX=40: DS=6: RB=2^DS
57 ME=SX*RB
60 DIM I(16,16), C(8)
64 D=12*4096: REM #C000 PER LA MATRICE D
85 PRINT "DIMENSIONI DELLA RAM=";ME; " BYTE"
88 GOSUB 1000: REM AZZERA LA MATRICE D
90 REM ** FASE DI ADDESTRAMENTO **
92 M$="ADDESTRAMENTO"
95 INPUT "QUALE CLASSE (1-8)? ";C
96 IF C<1 OR C>8 THEN 95
100 FOR I=1 TO 16: FOR J=1 TO 16: I(I,J)=0
101 NEXT J: NEXT I
110 REM ** ACCETTA L'IMMAGINE CREATA DALL'UTENTE **
120 GOSUB 2000: REM ROUTINE DI CREAZIONE DELL'IMMAGINE
```



CAROLINE CLAYTON

Gli operatori locali

I due diagrammi illustrano il particolare tipo di elaborazione di basso livello, usata dai sistemi "bottom-up", nel riconoscimento dell'immagine. L'"albero" a tre griglie sulla sinistra rappresenta gli "operatori locali", che esaminano i dati dell'immagine, registrano i valori limite ai bordi di passaggio e ne tracciano una rappresentazione sommaria usando linee rette. Gli operatori vengono fatti scorrere sui dati dell'immagine e ogni valore viene moltiplicato per il livello d'intensità del grigio dell'area osservata. I prodotti ottenuti generano valori limite solo quando è presente la caratteristica cercata. I nove valori contenuti nell'operatore locale sono parametri di contrasto e la loro configurazione indica le caratteristiche che determinano il riconoscimento dell'immagine



Scansione a 8 pixel

Il sistema di riconoscimento dell'immagine del "Wisard" si basa sul collegamento arbitrario tra gruppi di 8 pixel e registri a 8 bit. Ogni sequenza "ottupla" corrisponde a un banco di 256 locazioni di memoria. Durante la fase d'addestramento il valore contenuto nei registri rappresenta un indirizzo, all'interno del banco RAM, che dipende dalla configurazione dei pixel: nella locazione corrispondente viene depositato il valore 1. Se durante la fase di riconoscimento compare un'identica configurazione di pixel, viene generato lo stesso indirizzo del banco RAM e la presenza, in esso, del valore 1 determina il riconoscimento dell'immagine

```
130 R=RND(-1): REM VALORE CASUALE
140 FOR I=1 TO SX
150 A=RB*(I-1): REM CALCOLA INDIRIZZO DEL BANCO RAM
160 FOR J=0 TO (DS-1)
170 R1=INT (RND(1)*16+1):R2=INT(RND(1)*16+1)
190 A=A+I(R1,R2)*2^J
200 NEXT J
210 POKE (D+A),(PEEK(D+A) OR 2^(C-1))
220 NEXT I
230 INPUT "UN'ALTRA FASE DI ADDESTRAMENTO (S/N)?";A$
233 IF A$="S" THEN 90: REM RIPETE
234 REM ** FASE DI RICONOSCIMENTO **
235 M$="RICONOSCIMENTO"
240 FOR I=1 TO 16: FOR J=1 TO 16
244 I(I,J)=0
245 NEXT J: NEXT I
250 GOSUB 2000: REM ACCETTA UN'IMMAGINE DA CLASSIFICARE
270 R=RND(-1)
280 FOR C=1 TO 8: C(C)=0: NEXT C
290 FOR I=1 TO SX
300 A=RB*(I-1)
310 FOR J=0 TO DS-1
320 R1=INT(RND(1)*16+1): R2=INT(RND(1)*16+1)
340 A=A+I(R1,R2)*2^J
350 NEXT J
355 FOR C=1 TO 8
360 IF (PEEK(D+A)AND 2^(C-1))>0 THEN C(C)=C(C)+1
370 NEXT C
380 NEXT I
381 PRINT CHR$(147)
382 CX=0:X=0:Y=17:GOSUB 3000
385 FOR C=1 TO 8
388 PRINT "LA CLASSE "C" HA UN VALORE DI: C(C)/SX*100
390 IF C(C)>C(CX) THEN CX=C
400 NEXT C
404 PRINT "LA CLASSE PIU' SOMIGLIANTE E' LA N.";CX
410 INPUT "VUOI CLASSIFICARE UN'ALTRA IMMAGINE (S/N)? ";A$
420 IF A$="S" THEN 240: REM RIPETE
444 END
999 :
1000 REM ** INIZIALIZZA LA MEMORIA **
1010 FOR I=0 TO ME: POKE(D+1),0: NEXT I
1030 RETURN
1040 :
2000 REM ** ROUTINE DI CREAZIONE DELL'IMMAGINE **
2005 PRINT CHR$(147)
2010 X=0: Y=20: GOSUB 3000: PRINT "USARE A,B,S,D,X O
     LA BARRA SPAZIATRICE"
2012 PRINT "PER COMPORRE L'IMMAGINE"
2013 PRINT "FASE DI ";M$
2015 X=1:Y=1: GOSUB 3000
2020 T=0:A$=" ": H=0: V=1
2040 GET C$
2050 IF C$ = "A" THEN V = V - 1:IF V < 1 THEN V = 16
2060 IF C$ = "B" THEN V = V + 1:IF V > 16 THEN V = 1
2070 IF C$ = "S" THEN H = H - 1:IF H < 1 THEN H = 16
2080 IF C$ = "D" THEN H = H + 1:IF H > 16 THEN H = 1
2110 IF C$ = " " THEN T = 0:A$=C$
2120 IF C$ = "*" THEN T = 1:A$=C$
2130 I(H,V)=T
2140 X=H: Y=V: GOSUB 3000: PRINT A$;
2150 IF C$<>"X" THEN 2040
2155 GOSUB 2200
2160 RETURN
2170 :
2200 REM ** ROUTINE DI VISUALIZZAZIONE **
2210 FOR H=1 TO 16
2220 FOR V=1 TO 16
2230 X=H: Y=V: GOSUB 3000: IF I(H,V)=0 THEN PRINT":":
     GOTO 2240
2235 PRINT " ";
2240 NEXT V: NEXT H
2250 X=0: Y=17: GOSUB 3000
2260 RETURN
2270 :
3000 REM ** TABULAZIONE **
3010 PRINT CHR$(19); TAB(X); LEFT$(DN$,Y);
3020 RETURN
```

Basic & Basic

Spectrum:

Poiché lo Spectrum non possiede istruzioni AND e OR su singoli bit, occorre scrivere due brevi routine in codice macchina, qui contenute nel programma di caricamento in BASIC, che eseguano tali operazioni. Alla versione del programma di riconoscimento per Commodore 64 è necessario apportare le seguenti modifiche:

```
30 CLEAR 49999: GOSUB 4000
64 LET D = 50018
130 RANDOMIZE 1
210 LET X1 = PEEK(D + A): LET Y1 = 2↑(C—1): GOSUB 6000
215 POKE(D + A),R1
270 RANDOMIZE 1
360 LET X1 = PEEK(D + A): LET Y1 = 2↑(C—1): GOSUB 5000
365 IF R1>0 THEN C(C) = C(C) + 1
381 CLS
2005 CLS
3010 PRINT AT Y,X;
4000 FOR I = 50000 TO 50017
4010 READ A: POKE I,A: NEXT I
4020 RETURN
4030 DATA 62,0,14,0,161,6,0,79,201
4040 DATA 62,0,14,0,177,6,0,79,201
5000 REM *** AND ***
5010 POKE 50001,X1: POKE 50003,Y1
5020 LET R1 = USR 50000: RETURN
6000 REM *** OR ***
6010 POKE 50010,X1: POKE 50012,Y1
6020 LET R1 = USR 50009: RETURN
```

Il Magic Mouse

Il primo dispositivo progettato per funzionare sul Commodore 64

Se si considera il successo dei sistemi operativi basati su mouse, è difficile credere che la loro introduzione abbia inizialmente suscitato forti contrasti tra le case produttrici di microcomputer. Il relativo insuccesso del Lisa della Apple, il primo micro basato su un SO di questo tipo, ha in parte confermato le perplessità del pubblico nell'accettare sistemi basati su mouse.

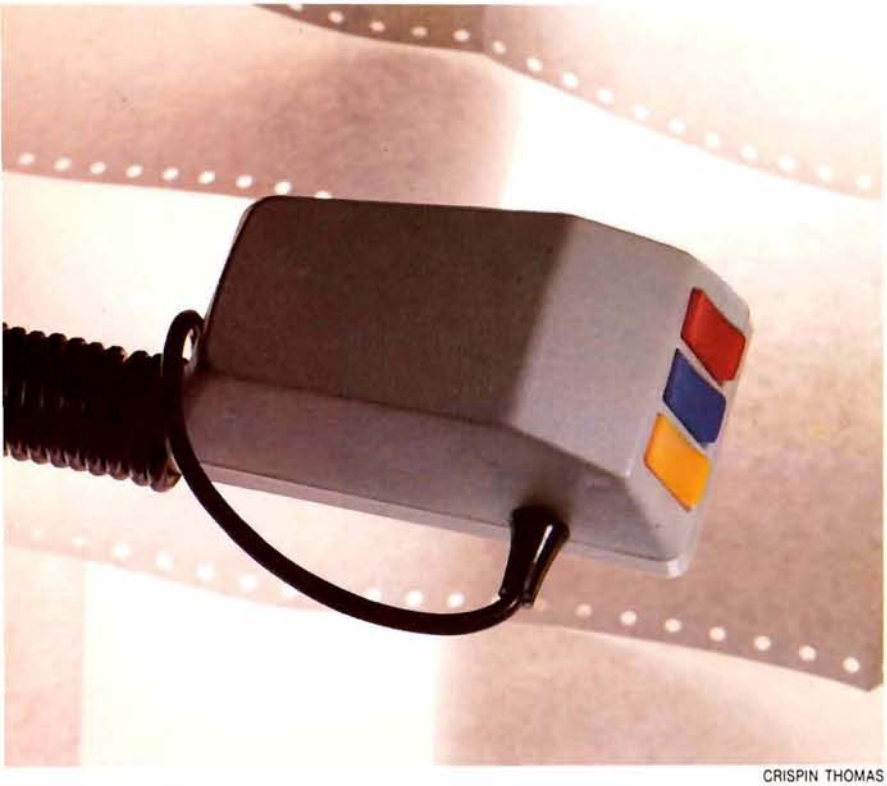
Tuttavia, il successo ottenuto in seguito dal Macintosh ha contribuito ad abbattere molte riserve, oltre ad accrescere la fiducia della Apple verso tale sistema. Oggi molte industrie di computer competono nella produzione di macchine basate su mouse e molte compagnie indipendenti sviluppano mouse corredati di software per gli apparecchi esistenti.

Il Magic Mouse della SMC Supplies è stato uno dei primi package di questo genere creati per il Commodore 64. Il sistema si compone di un mouse, corredato da relativo software (su cassetta e dischetto), che contiene quattro pacchetti applicativi. Le dimensioni del mouse SMC (125x66x50 mm.) sono circa il doppio di quelle del mouse fornito dalla Apple. Inoltre il Magic Mouse possiede anteriormente 3 pulsanti colorati mentre la maggior parte dei mouse ne ha solo due. Questi pulsanti servono per selezionare funzioni che, ovviamente, variano a seconda della particolare applicazione. All'interno dell'apparecchio c'è una sfera di gomma rigida a diretto contatto con una coppia di codificatori angolari ad albero.

Il progetto originale prevedeva l'impiego di una sfera d'acciaio, ma questa scelta si è dimostrata inadeguata e il lancio del pacchetto è stato rinviato per consentirne la sostituzione. La versione definitiva del mouse funziona in modo perfetto.

Un topo... magico

Il Magic Mouse della SMC è stato uno dei primi sistemi basati su mouse per il Commodore 64. Sebbene non sia un vero e proprio pacchetto WIMP (Window, Icon, Mice, Programming), esso può essere usato con buoni risultati come "strumento" per sviluppare programmi. Il mouse si innesta alla porta del joystick del Commodore 64 e le icone sullo schermo vengono selezionate usando i tre pulsanti colorati



CRISPIN THOMAS

Programmi di utilità

Nel suo complesso, il Magic Mouse non dispone di un "ambiente" operativo come quello del Macintosh: si tratta piuttosto di un pacchetto per creazioni grafiche. Come si è detto, il mouse è corredato da quattro programmi: l'*Hi-Res Designer*, lo *Sprite Designer*, l'*Icon Designer* e il *Mouse Controller*.

Per eseguire correttamente le diverse applicazioni, occorre innanzi tutto caricare in memoria il file di sistema e "tarare" il mouse. Mentre il file di sistema su dischetto viene caricato automaticamente assieme a un menu principale per la selezione dei quattro programmi di utilità, quello su cassetta presenta alcune differenze. Infatti il file di sistema fa parte dell'*Hi-Res Designer* e questo programma deve essere caricato in memoria prima di poter selezionare le altre applicazioni. Inoltre il cursore deve essere posizionato nell'angolo in basso a sinistra dello schermo prima di iniziare il lavoro.

L'*Hi-Res Designer* è un programma "tavolozza" simile al software del Koala Pad (vedere a pag. 1109). Quando viene caricato, sullo schermo compaiono una serie di riquadri contenenti differenti funzioni come Box, Draw o Fill, selezionabili posizionando il cursore sul riquadro corrispondente e premendo il pulsante rosso del mouse.

Analogamente, per scegliere il colore corrente di primo piano o di sfondo, si sposta il cursore su una delle 16 linee di colore disponibili. Infine, per visualizzare la "tela" da disegno, si preme il pulsante blu.

I pacchetti *Icon* e *Sprite Designer* sono piuttosto simili, ma producono risultati assai diversi. Come molti programmi del suo genere, lo *Sprite Designer* visualizza una griglia standard di 24 colonne per 21 righe. I pixel vengono accesi posizionando il cursore nel punto desiderato e premendo il pulsante rosso. Lo sprite creato compare in una finestra aperta nell'angolo in alto a sinistra dello schermo e, quando il cursore viene allontana-

to dalla griglia, nell'angolo in basso a destra compaiono una serie di utili opzioni tra le quali: l'espansione dello sprite, il cambiamento del suo colore o la scelta di un altro sprite.
Il programma *Icon Designer* visualizza la griglia in un angolo e l'icona in una finestra aperta nella parte superiore dello schermo ma si differenzia dal precedente perché viene usato per creare immagini definite dall'utente (come ad esempio un nuovo set di caratteri), che possono essere salvate e utilizzate in seguito.
Il *Mouse Controller*, infine, non svolge specifiche funzioni grafiche ma fornisce al sistema il software di controllo necessario per elaborare le realizzazioni grafiche dell'utente. A prima vista può sembrare il pacchetto meno utile ma, in realtà, è quello che possiede maggiori potenzialità poiché permette all'utente di scrivere programmi che utilizzano il mouse.
Il principale vantaggio offerto dai sistemi a mouse, rispetto ai digitalizzatori convenzionali, è l'affidabilità del cursore, che permette di posizionare con notevole precisione le linee e i colori nel disegno. I digitalizzatori convenzionali, infatti, sono basati su una griglia di nodi tra loro collegati, la cui risoluzione è decisamente inferiore a quella dello schermo: se la "penna" si trova tra due nodi, il computer non è in grado di scegliere una posizione intermedia, cosicché tende a selezionare a caso l'una o l'altra. Tale imprecisione deteriora, talvolta in modo molto evidente, l'immagine progettata dall'utente. Nei sistemi dotati di mouse, invece, il cursore risponde solo ai cambiamenti indotti nei potenziometri dalla rotazione della sfera, restando fermo se il mouse non viene spostato.
Tuttavia niente è perfetto, neanche il Magic Mouse: alcune funzioni dell'*Hi-Res Designer* sono meno comode da usare rispetto ad altri pacchetti. Il comando RUB, ad esempio, permette di cancellare gli errori solo un pixel alla volta.
Questo processo è piuttosto lento e spesso porta a commettere nuovi errori durante il processo di cancellazione.
Una tecnica molto più comoda sarebbe quella di eliminare tutti i dettagli aggiunti all'immagine dall'ultima selezione del menu grafico: questa operazione causa la perdita di parti dell'immagine prive d'errori ma permette di riprendere il lavoro senza dover eseguire laboriose correzioni.
Il manuale del Magic Mouse ha ottime qualità didattiche: descrive in modo esauriente il funzionamento di ogni pacchetto e fornisce, oltre ad alcuni semplici programmi campione, anche informazioni sulla programmazione sufficienti per consentire all'utente di introdurre le immagini create nel proprio software.
Sebbene il Magic Mouse della SMC non sia paragonabile all'Emulator del Macintosh, il software è stato sviluppato con l'intento di offrire ai programmatori uno strumento di lavoro simile al mouse AMX (vedere a pag. 1530) o al programma *DR GEM*.



Per concessione della DIMENSIONS GRAPHICS

JOHN CLEMENTSON

Creazioni artistiche

Queste immagini sono state realizzate con il programma applicativo "Hi-Res Designer" e il Magic Mouse della SMC. Il programma permette di adoperare i comandi abitualmente disponibili sui pacchetti di questo tipo (ad esempio: LINE, DRAW, FILL e CIRCLE). Un'immagine può contenere un massimo di 16 colori. Infine, il programma offre tutta una serie di "pennelli" che consentono di variare l'ampiezza e la trama delle linee

MAGIC MOUSE

DIMENSIONI

mm. 125 x 66 x 50

INTERFACCIA

Il Magic Mouse viene innestato nella porta per joystick del Commodore 64

SOFTWARE

Il mouse è corredato di quattro programmi applicativi, registrati sia su cassetta che su dischetto

DOCUMENTAZIONE

Il manuale fornisce ampie informazioni su come predisporre l'uso del mouse, utilizzare i programmi in dotazione e includere i risultati nel proprio software

PRO

Il mouse è realizzato in modo molto robusto e affidabile. Non si dovrebbero incontrare difficoltà nel trasferire gli sprite e la grafica definita da utente nei propri programmi

CONTRO

L'intero sistema, per le sue stesse caratteristiche, non può né deve essere paragonato ad altri, molto più sofisticati, che prevedono l'impiego di una maggior quantità di memoria e di sistemi operativi più complessi

Liste attive

La serie dedicata al LISP si conclude con alcune nuove considerazioni sulle liste

Per ottenere una notazione uniforme, che consenta di manipolare con facilità identificatori, caratteri, numeri e liste, il LISP ricorre a "puntatori".
Questi possono essere agevolmente simulati in qualsiasi linguaggio di programmazione, compreso il BASIC. Ad esempio, se si eseguono le seguenti istruzioni:

```
POKE 100,55: POKE 99,100 (sulla maggior parte dei micro)
```

oppure

```
?100=55: ?99=100 (su Micro BBC e Electron)
```

le locazioni 100 e 99 dovrebbero contenere i valori in esse depositati: è quindi possibile usare il contenuto della locazione 99 come un puntatore alla locazione 100. In questo caso, i comandi:

```
PRINT PEEK (PEEK(99))
```

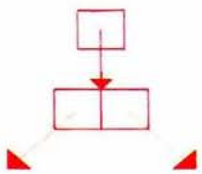
oppure

```
PRINT ?(?99)
```

danno come risultato 55 (si noti che, su alcuni micro, le locazioni 99 e 100 possono trovarsi in ROM e non devono essere modificate).
Naturalmente è possibile creare puntatori per qualsiasi locazione di memoria ma, per quelle superiori a 255, occorrono due byte, affinché possano contenere indirizzi compresi tra 0 e 65535. Supponiamo che il valore 55 punti, a sua volta, ad un'altra locazione: tale sequenza di puntatori compone una lista con la seguente struttura:

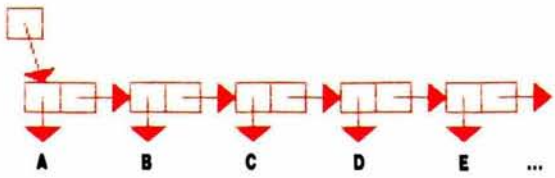


Purtroppo il modulo precedente è di scarsa utilità, poiché la lista non contiene informazioni. Ma queste possono essere introdotte combinando le locazioni in modo da assegnare due puntatori a ciascun nodo.



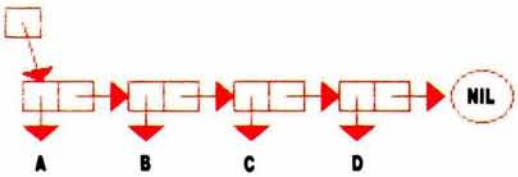
Adesso possiamo usare il modulo per rappresentare la lista "infinita":

(A B C D E ...)



Come accennato in precedenza (vedere a pag. 1864), per porre un termine alle liste infinite occorre introdurre un puntatore finale che faccia riferimento a una lista vuota (o costruzione NIL). In questo modo si ottiene la lista "finita":

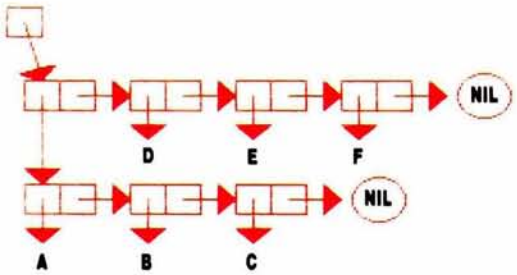
(A B C D)



Una lista che contenga a sua volta una lista indica semplicemente che uno dei puntatori di sinistra punta a una nuova lista e non, come nel caso precedente, a un semplice atomo. Quindi la lista:

((A B C) D E F)

può essere rappresentata con il modulo:



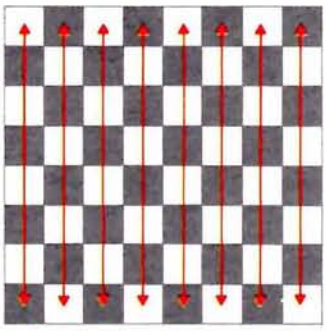
Questo esempio chiarisce anche lo scopo delle funzioni CAR e CDR. CAR (la "testa" del suo argomento) è il puntatore di sinistra del primo nodo del suo argomento, mentre CDR (tutti gli elementi eccetto la testa del suo argomento) è il puntatore di destra del primo nodo. Quindi, se si chiama L la lista finita, l'istruzione:

```
CONS((CAR L) (CDR L))
```

fornisce come risultato la lista originale L. Quindi CONS è una procedura per la costruzione di liste che inserisce i suoi due argomenti in una nuova lista e fornisce come risultato un puntatore relativo al primo elemento della nuova lista. Poiché questa tecnica permette di creare qualsiasi tipo di lista, ma richiede numerose chiamate di CONS, il LISP dispone di una funzione più concisa: LIST. È però sempre preferibile usare CONS per inserire una nuova informazione alla testa di una lista.
Quando si aggiunge la coda di una lista, è necessario creare una copia della lista originale modificando però il suo puntatore finale in modo che punti al nuovo elemento introdotto.
Alcune versioni del LISP possiedono un'istruzione (APPEND) che esegue tale modifica automaticamente, anche se in modo meno efficiente di CONS.
Se il programma richiede un'alta velocità d'esecuzione, si può eliminare la creazione della copia usando un'istruzione che modifica il puntatore finale direttamente nella lista originale.
Esaminiamo adesso un esempio pratico di programmazione relativo a un famoso problema di scacchi. Il giocatore deve posizionare otto regine su una scacchiera in modo che nessuna possa attaccare le altre. Per risolvere il problema le regine devono occupare differenti linee e colonne, senza però trovarsi su una stessa diagonale.

L'"Enigma delle 8 Regine"

Supponiamo che ogni colonna della scacchiera contenga una regina. La posizione di ciascuna regina (1 - 8) viene inserita in una lista chiamata REGINE. La riga di ciascuna regina (indicata dal valore registrato nella lista) può variare e viene calcolata dal LISP.



REGINE (Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8)

All'inizio del programma la lista REGINE è vuota. Ad ogni passaggio viene inserita una nuova regina sulla riga 1 all'inizio della lista e la sua posizione viene modificata (funzione MODIFICA), assieme a quella delle altre, finché tutte le regine raggiungono una casella "sicura". A questo punto la lista diviene la nuova lista REGINE.
Il processo termina quando la lista contiene otto regine: a questo punto ST_SCACC visualizza la lista finale delle otto regine.

La procedura iniziale, chiamata RISOLVE, ha il seguente formato:

```
(DEFUN RISOLVE ( )
  (SETQ REGINE ( ))
  (LOOP (UNTIL EQ (LUNG (REGINE) 8))
    (SETQ REGINE (MODIFICA (CONS 1 REGINE))))
  (ST_SCACC '(QR QN QB Q K KB KN KR) REGINE))
```

La funzione RISOLVE, che non possiede argomenti, crea la lista vuota REGINE. Quindi la funzione LOOP assegna a REGINE il nuovo valore della lista modificata (MODIFICA) di una regina nella riga 1, seguito dalla lista precedente. Il ciclo iterativo si conclude quando la lista REGINE contiene 8 elementi. Infine viene visualizzata la lista finale.
Nella procedura RISOLVE compaiono tre nuove funzioni. LUNG, che fornisce il numero di elementi contenuti nel suo argomento, viene definita nel seguente modo:

```
(DEFUN LUNG (L (N))
  (SETQ N 0)
  (LOOP (WHILE L N)
    (SETQ N (ADD1 N))
    (SETQ L (CDR L)) ))
```

Il LISP della Acornsoft usa l'argomento N, posto tra parentesi tonde, per rappresentare gli argomenti facoltativi. Nel nostro caso, N non viene mai fornito alla funzione e opera come una variabile locale.
La seconda funzione, ST_SCACC, combina gli elementi delle sue due liste di argomenti per produrre un risultato visualizzabile e comprensibile. Questa funzione viene definita nel seguente modo:

```
(DEFUN ST_SCACC (RIGHE COLONNE)
  (COND ((NULL RIGHE) T)
    (T (PRINTC (CAR RIGHE) (CAR COLONNE))
      (ST_SCACC (CDR RIGHE) (CDR COLONNE)) )))
```

Presumendo che entrambe le liste abbiano la stessa lunghezza, la funzione si conclude se la prima condizione è vera (una lista è vuota). In caso contrario, viene visualizzato il primo elemento di ciascuna lista (seguito da un ritorno carrello) e la funzione richiama se stessa per operare sui successivi elementi di ciascuna lista.
La terza funzione, MODIFICA, che esegue in pratica tutto il lavoro, è così definita:

```
(DEFUN MODIFICA (REGINE)
  (COND ((EQ (CAR REGINE) 9)
    (MODIFICA (CONS (ADD1 (CADR REGINE))
                    (CDDR REGINE) )))
    ((SICURA REGINE) REGINE)
    (T (MODIFICA (CONS (ADD1 (CAR REGINE))
                       (CDR REGINE) )))))
```

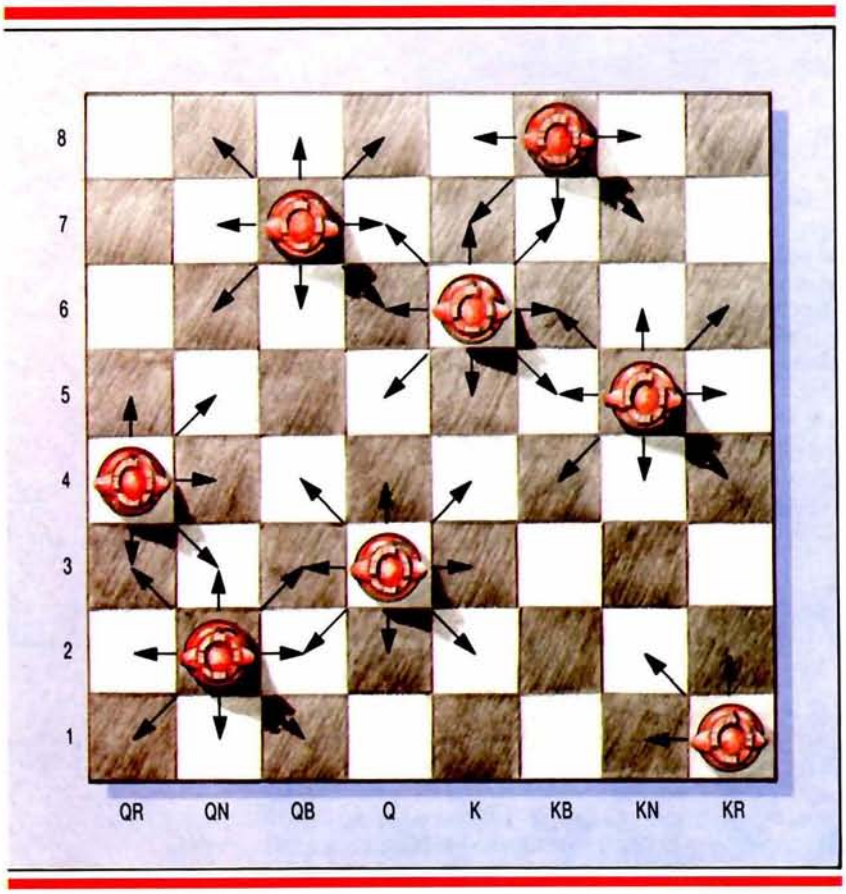
La funzione MODIFICA verifica innanzi tutto se l'ultima regina esaminata è stata collocata al di fuori del margine superiore della scacchiera (riga 9). In questo caso occorre muovere una o più regine disposte in precedenza.
Si ricordi che questa regina parte dalla riga 1 e che la sua posizione viene confrontata con

quella di tutte le altre regine (dalla terza condizione della funzione MODIFICA). Nel caso in cui la disposizione dei pezzi sulla scacchiera ne impedisca la collocazione, la procedura MODIFICA chiama se stessa e interviene sul resto della lista REGINE (ossia su tutte le altre regine eccetto l'ultima inserita).
Eseguite queste operazioni, la funzione MODIFICA incrementa la riga dell'ultima regina. Tale procedimento, detto "spostamento di ritorno", è comunemente impiegato negli algoritmi di questo tipo. Se la seconda condizione risulta vera, tutte le regine sulla scacchiera occupano una casella SICURA e il programma visualizza la lista finale. In caso contrario, la posizione dell'ultima regina inserita viene incrementata e confrontata con quella di tutte le altre.
Resta da definire la funzione SICURA, che deve verificare se tutte le regine occupano una casella sicura. Considerato che la posizione all'interno della lista definisce la colonna occupata e che due regine non possono avere la stessa posizione di lista, non è necessario confrontare le intersezioni delle colonne, ma solo quelle delle righe e delle diagonali. Supponiamo di usare una funzione NO_VERIF che dia un risultato "vero" se ogni regina occupa una casella "sicura". In tal caso, la funzione SICURA può essere così definita:

```
(DEFUN SICURA (REGINE)
  (COND ((NULL REGINE) T)
    (T (AND (NO_VERIF (CAR REGINE)
                      (CDR REGINE) 1)
      (SICURA (CDR REGINE)) ))))
```

Un problema... regale

L'Enigma delle 8 Regine può essere elegantemente risolto utilizzando le funzioni per la manipolazione delle liste e le procedure ricursive del LISP. Prima di esaminare la soluzione presentata, si provi a disporre 8 pezzi su di una scacchiera (assegnando loro il valore di regina) in modo che nessuno di essi si trovi in posizione di pericolo. Negli scacchi, la regina può muoversi e catturare pezzi in linea retta (orizzontalmente, verticalmente o diagonalmente) per un qualsiasi numero di caselle



La funzione SICURA ha il seguente significato: "La prima regina della lista REGINE non si trova su una riga o una diagonale già occupata dalle precedenti regine e (AND) tutte le regine del resto della lista occupano una casella sicura".
Definiamo adesso la funzione NO_VERIF, che produce un risultato vero se non riscontra intersezioni lungo le righe o le diagonali:

```
(DEFUN NO_VERIF (NUOVA RESTO COL)
  (COND ((NULL RESTO) T)
    (T (AND (NOT (EQ NUOVA (CAR RESTO)))
(NOT (EQ COL
(ABS (DIFFERENZA NUOVA (CAR RESTO)))))
(NO_VERIF NUOVA (CDR RESTO) (ADD1 COL))))))
```

NUOVA rappresenta l'ultima regina inserita mentre RESTO è la lista delle altre regine sulla scacchiera. Supponendo che la scacchiera contenga altre regine (NULL RESTO è falso), la funzione AND è vera se:

1) L'ultima regina inserita non occupa la stessa riga della prima regina del resto della lista.

2) Le regine non si trovano sulla stessa diagonale, ossia la differenza di riga, fornita dalla funzione ABS, non coincide con la differenza di colonna (COL) dove COL contiene il numero di colonne, inizialmente 1, che separano la regina in esame dalla successiva regina della lista.
   La funzione ABS è già stata definita a pag. 1865:

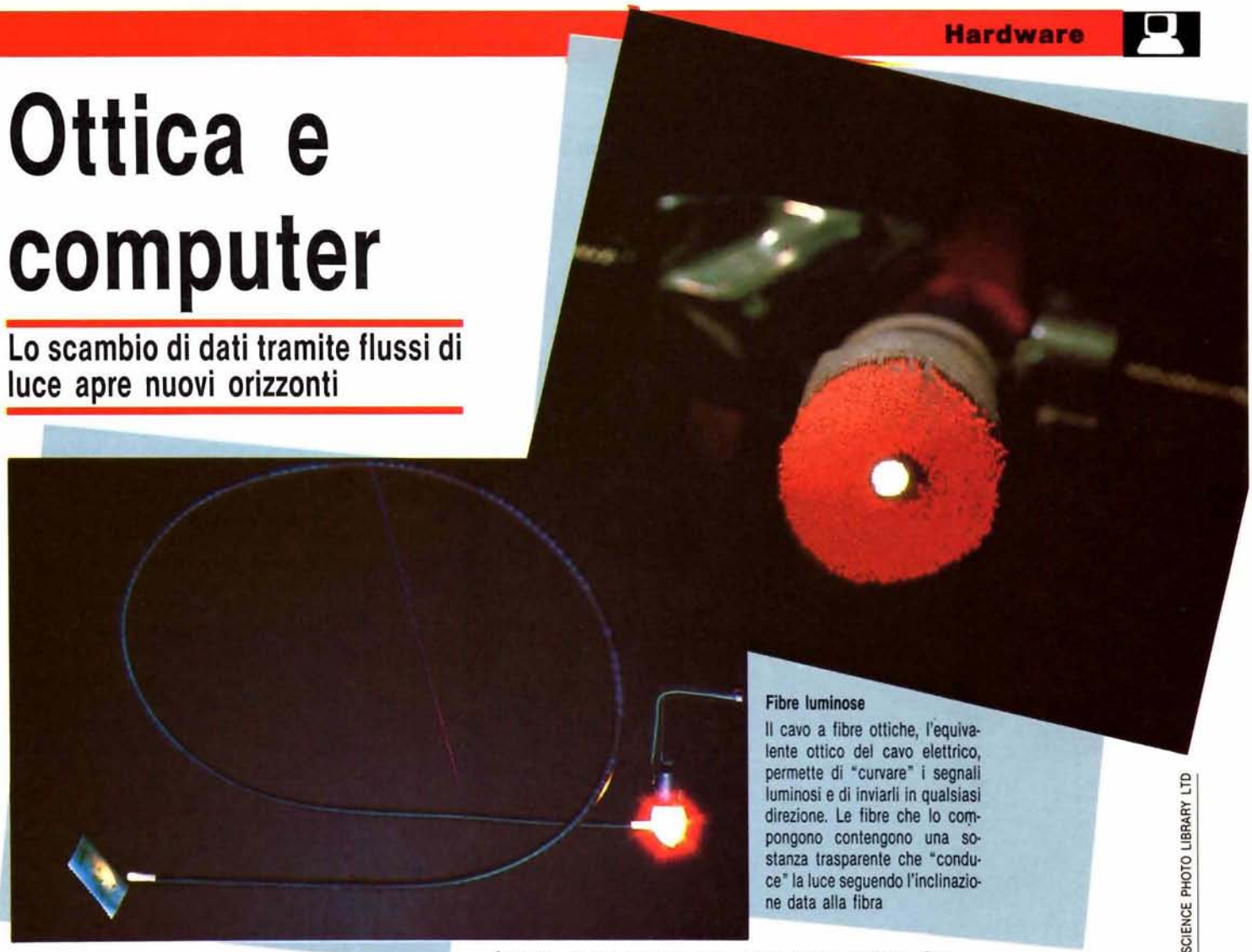
```
(DEFUN ABS (NUMERO)
  (COND ((MINUSP NUMERO) (MINUS NUMERO))
    (T NUMERO) ))
```

3) I punti 1 e 2 sono veri sia per la regina in esame sia per la successiva nella lista. Questa verifica viene eseguita dalla funzione che chiama se stessa con qualsiasi elemento della lista RESTO, eccetto il primo, e incrementa la differenza di colonna (COL).

A questo punto il programma è completo e può essere eseguito: la figura in basso mostra la soluzione del problema. Si ricordi che questo listato è stato scritto usando il LISP Acornsoft, ma sono sufficienti piccole modifiche per adattarlo ad altri dialetti. La procedura che genera lo "spostamento di ritorno" può sembrare piuttosto complessa: per seguirne più facilmente lo sviluppo si può far visualizzare la lista REGINE, all'interno della funzione RISOLVE, prima di ogni chiamata per la funzione MODIFICA.
La tecnica di programmazione "top-down" applicata all'"Enigma delle 8 Regine" mostra come il procedimento "da funzione a funzione" faciliti la definizione dei programmi LISP. In questo caso, ad esempio, la funzione SICURA, all'interno di MODIFICA, viene invocata ancor prima di definirla: ciò offre un'ulteriore dimostrazione della grande versatilità del linguaggio LISP.

Ottica e computer

Lo scambio di dati tramite flussi di luce apre nuovi orizzonti



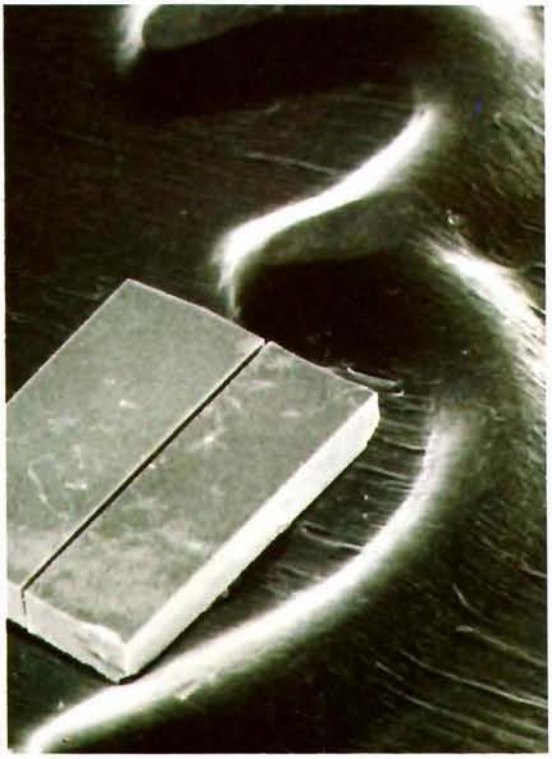
Fibre luminose
Il cavo a fibre ottiche, l'equivalente ottico del cavo elettrico, permette di "curvare" i segnali luminosi e di inviarli in qualsiasi direzione. Le fibre che lo compongono contengono una sostanza trasparente che "conduce" la luce seguendo l'inclinazione data alla fibra

Per concessione della SCIENCE PHOTO LIBRARY LTD

Quando si pensa ad un computer, abitualmente si immagina una macchina basata su interruttori elettronici binari, capace di memorizzare ed elaborare informazioni. Tuttavia, sarebbe sbagliato concludere che questo sia l'unico tipo di apparecchio realizzabile. In teoria un computer può usare qualsiasi metodo fisico d'immagazzinamento delle informazioni: cariche elettriche, ingranaggi meccanici, nastri di carta perforati, perline, sassi, conchiglie e via dicendo.
Almeno per il momento però, il miglior dispositivo di memoria inventato è il "semiconduttore" elettronico, poiché consuma poca energia, ha dimensioni microscopiche e, soprattutto, è molto rapido. La velocità di elaborazione di un computer dipende dal tempo necessario per commutare, da una posizione all'altra, le celle di memoria a informazioni binarie (o "interruttori" logici) e dalla velocità nello scambio di informazioni tra tali interruttori. Benché i circuiti dei computer che usano flussi di elettroni eseguano queste operazioni con grande rapidità, è teoricamente e tecnicamente possibile fabbricare circuiti ancora più veloci, basati sullo scambio di segnali luminosi.
La ricerca nel campo dell'elaborazione ottica e delle discipline collegate, quali la "logica fotonica", è assai avanzata e già sono disponibili molti dei componenti che verranno in futuro incorporati nei computer ottici. Le fibre ottiche, ad esempio, vengono comunemente impiegate per trasmettere segnali telefonici e televisivi. Inoltre sono stati sviluppati sistemi per la propagazione pilotata delle onde, analoghi alle piste ramate delle schede a circuiti stampati, ed è in fase di sperimentazione il dispositivo centrale di questi nuovi computer, ossia l'equivalente ottico dell'interruttore elettronico binario.
La progettazione dell'interruttore binario ottico segue due direttrici principali: l'interruttore elettro-ottico e l'interruttore interamente ottico. Dei due, il primo a essere usato sarà probabilmente l'ibrido elettro-ottico, che verrà abbinato a componenti elettronici attualmente in uso. Entrambi gli interruttori si basano sul fenomeno dell'interferenza luminosa, controllata da cambiamenti dell'indice di rifrazione in un materiale ottico non lineare (di solito una lega di gallio o di litio). L'indice di rifrazione di un materiale è il rapporto tra la velocità della luce nel vuoto e quella all'interno del materiale. Se l'indice di rifrazione di un particolare tipo di cristallo è, ad esempio, 1,5 allora, poiché la luce viaggia nel vuoto a circa 300.000 Km. al secondo, la velocità all'interno del cristallo sarà di circa 200.000 Km. al secondo.
Il funzionamento dell'interruttore binario di tipo elettro-ottico poggia su un dispositivo chiamato "interferometro di Mach Zehnder",

La sfida ottica

Un altro componente fondamentale per la costruzione dei computer ottici è un dispositivo semiconduttore che possa essere regolato in modo da produrre una gamma di frequenze luminose pure. Questo strumento è indispensabile per realizzare scambi di comunicazioni basate su segnali luminosi di lungezza d'onda e frequenza standard



Per concessione della SCIENCE PHOTO LIBRARY LTD

basato su un principio fisico molto semplice: un fascio luminoso, generato da una sorgente, viene suddiviso in due raggi che attraversano due canali di materiale non lineare e vengono successivamente ricombinati. Se a nessuno dei due canali dell'interferometro viene applicata una corrente, i due raggi in uscita sono ancora "in fase" e generano un fascio luminoso forte (stato "on"). Se invece l'indice di rifrazione di un canale viene modificato mediante un segnale elettrico, i due raggi in uscita non si trovano più "in fase" (uno dei due ha viaggiato più velocemente), pertanto il raggio ricombinato risulta molto più debole (stato "off").

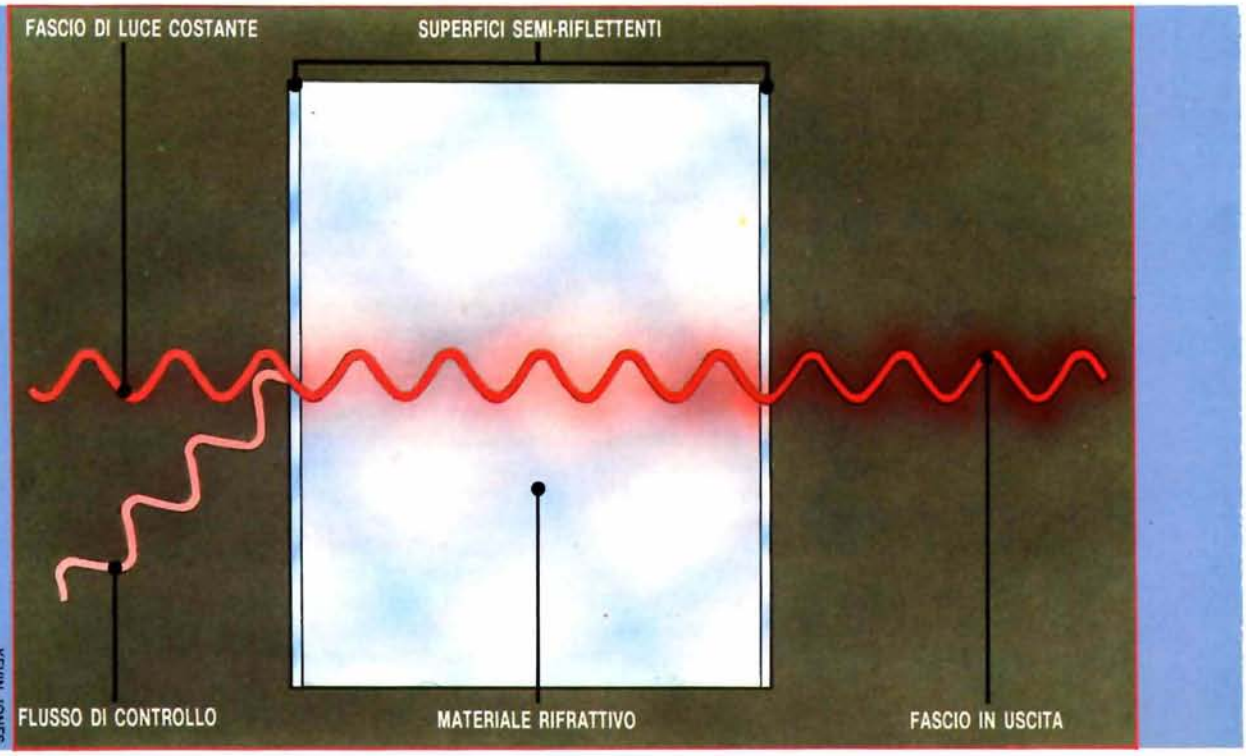
Questo tipo d'interruttore può costituire la base di un semplice circuito integrato ottico. Il circuito si dimostra più veloce del suo equivalente elettronico perché la luce (grazie alla più alta frequenza delle onde luminose) trasporta l'informazione ad una velocità circa 10.000 superiore rispetto a quella dei segnali elettronici. Nonostante la superiorità delle sue prestazioni, la velocità di un simile circuito risulta in parte rallentata dal processo di commutazione che è controllato da un "feedback" di tipo elettrico. Tuttavia, usando la luce per operare la commutazione (ossia un fascio di luce debole per controllarne uno più forte) è possibile ottenere velocità maggiori.

Il conceto di base è praticamente identico a quello del transistor elettronico. Il prototipo di un interruttore interamente ottico, chiamato "transfasore", è già stato realizzato da un'equipe di scienziati alla "Heriot Watt University" di Edimburgo. Lo strumento è capace di eseguire una commutazione in appena un picosecondo (10-12 secondi), mentre il più veloce transistor elettronico svolge la stessa operazione in un nanosecondo (10-9 secondi). Per questo motivo i futuri computer interamente ottici potranno facilmente funzionare a una velocità di ben 1.000 volte superiore rispetto ai computer elettronici.

Il transfasore si basa sul progetto di un interferometro, originariamente ideato dai fisici francesi Charles Fabry e Alfred Perot nel 1896. Una lamina di materiale non lineare viene interposta tra due superfici riflettenti. Quando un fascio di luce attraversa una superficie, viene riflesso avanti e indietro all'interno del materiale interposto prima di uscire dalla superficie opposta. L'interazione reciproca tra le onde luminose "intrappolate" dipende dall'indice di rifrazione del materiale non lineare, indice che può essere modificato tramite un fascio luminoso di controllo. Se l'indice di rifrazione è tale da permettere alle

Scontro tra... micro-giganti

Il componente base dei computer elettronici è il transistor, mentre quello del computer ottico è il cosiddetto "transfasore". Un fascio di luce costante viene introdotto da un'estremità del materiale, assieme ad un flusso luminoso "di controllo". All'interno del dispositivo i due fasci luminosi interferiscono tra loro: una lieve variazione del flusso di controllo produce notevoli balzi d'intensità nel fascio di luce in uscita. Questa caratteristica consente di adoperare il transfasore come dispositivo di commutazione logica



KEVIN JONES

onde luminose di rimanere in fase e di rinforzarsi a vicenda, viene emesso un fascio di luce di forte intensità. In caso contrario, il fascio in uscita è debole. Queste due condizioni sono alternative e corrispondono alle posizioni "on" e "off".

Vantaggi teorici

In sintesi, il successo dei computer ottici dipenderà dagli effettivi vantaggi teorici e tecnici che essi offriranno rispetto ai computer convenzionali. Sulla carta, un computer ottico è per molti aspetti superiore a quello elettronico. Si consideri, ad esempio, la velocità di commutazione. Perché un transistor possa funzionare da interruttore, è necessario che la sua base sia attraversata da un flusso di elettroni. Tuttavia la velocità massima degli elettroni all'interno di un semiconduttore e le dimensioni minime della base di un transistor presentano limiti invalicabili (ovviamente, più piccola è la base, minore è il percorso che gli elettroni devono compiere). Quindi un computer ottico, che trasmette informazioni alla velocità della luce, è intrinsecamente più veloce.

Un secondo fattore che limita i computer convenzionali è il cosiddetto "collo di bottiglia di Von Neumann". Questa è una caratteristica dell'architettura di base comune a quasi tutti i computer e riguarda la procedura "parola dopo parola" (di tipo "seriale") usata per accedere alle informazioni in memoria. Un computer ottico consente invece di accedere alla memoria "in parallelo" perché, contrariamente ai segnali elettrici, i fasci luminosi non devono essere isolati. Ciò consente di convogliare più fasci luminosi indipendenti sullo stesso transistor ottico e di commutarli contemporaneamente.

Il terzo limite dei computer convenzionali dipende dallo scarso spazio di registrazione disponibile su un chip piano. Generalmente il numero di collegamenti realizzabili su un singolo chip è rigorosamente limitato dal cosiddetto "fattore di carico massimo d'uscita" dei piedini, che determina il numero di piedini applicabili a una lastrina di silicio. Impiegando particolari tecniche olografiche tridimensionali, è possibile, *teoricamente* introdurre una maggior libertà di cablaggio e modificare istantaneamente la configurazione delle connessioni.

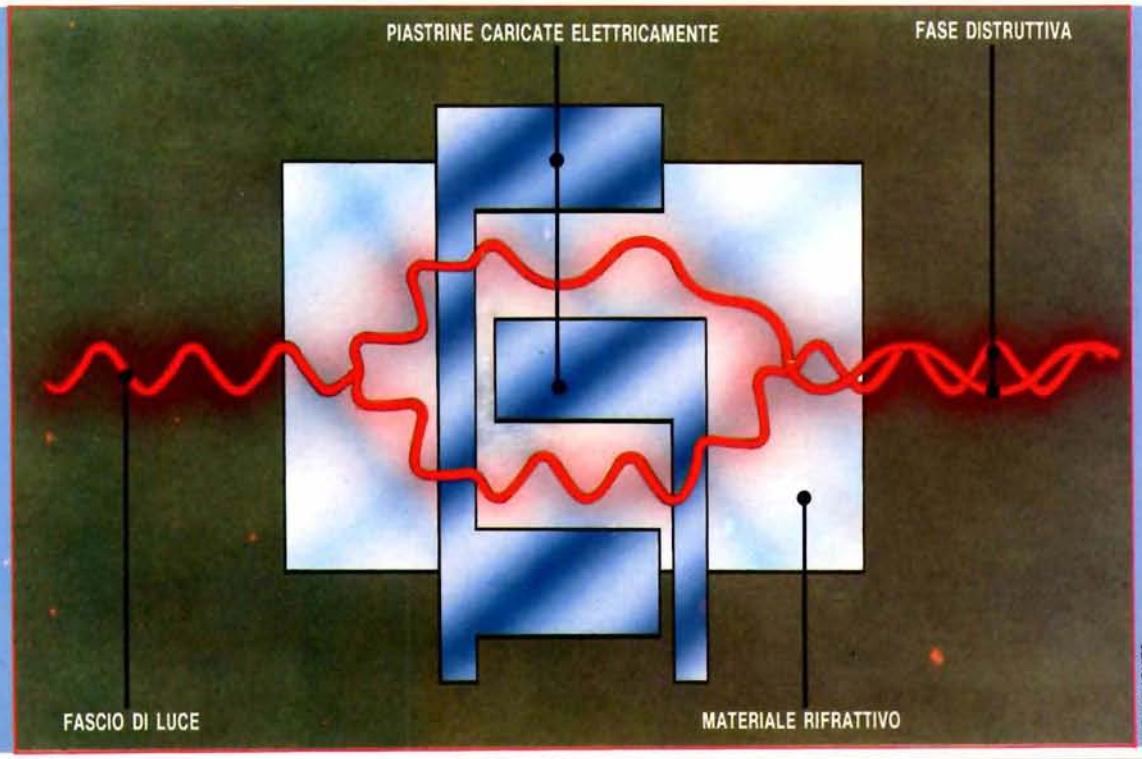
Infine i computer ottici offrono notevoli vantaggi pratici nel collegamento con le periferiche. Le reti a fibre ottiche, che già vengono usate per le comunicazioni tra apparecchi elettronici, sono estremamente rapide e consentono di trasmettere senza interferenze più messaggi luminosi su una stessa fibra. I computer ottici possono collegarsi direttamente a queste reti senza l'intervento di ulteriori interfacce opto-elettriche.

In base a queste considerazioni, non dovrebbe essere lontano il giorno in cui i computer ottici faranno la loro comparsa sul mercato. Tuttavia, i vantaggi teorici delle tecniche ottiche rispetto a quelle elettroniche sono destinati a scontrarsi con solide considerazioni di ordine pratico. Infatti i travolgenti successi ottenuti nella miniaturizzazione dei chip contrastano nei fatti la comparsa dei computer ottici.

Il numero di transistor assemblabili su un chip cresce di giorno in giorno e si prevede che nel 2000 i chip potranno contenerne da 10 a 100 milioni.

Dall'"integrazione su larga scala" (LSI) si è passati all'integrazione su scala "molto larga" (VLSI) e "ultra larga" e dal primo "processore parallelo" (vedere a pag. 1708) a processori paralleli di potenza sempre maggiore.

Infine, il continuo sviluppo e perfezionamento delle più recenti tecnologie elettroniche ad alta velocità rende ancor più incerto il futuro dei calcolatori ottici.



KEVIN JONES

Fase contro fase

L'interferometro di Mach Zehnder combina l'azione di dispositivi elettronici ed ottici. La luce che entra da un'estremità del dispositivo viene suddivisa in due fasci. Variando l'indice di rifrazione, limitatamente ad uno dei due fasci, la velocità di questo risulta diversa dalla velocità dell'altro. Tale diversità influisce sui due segnali modificandone la fase. Se all'uscita le due fasi si trovano in opposizione, il segnale risulta "nullo" (livello logico 0)

Hi-Fi con la MIDI

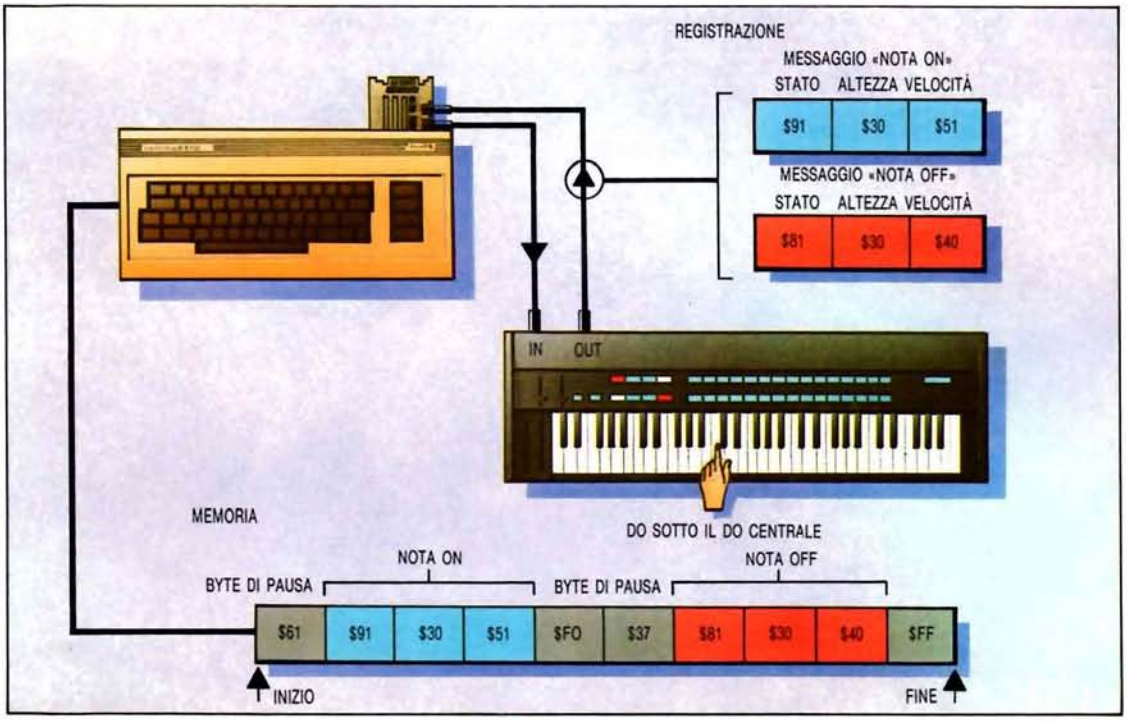
Lo sviluppo di un programma che fa del computer un registratore in tempo reale

Quando vengono premuti i tasti di una tastiera MIDI oppure viene azionato un dispositivo di controllo quale una manopola di regolazione della tonalità, i segnali generati vengono inviati alla presa MIDI OUT, posta sul retro del sintetizzatore, sotto forma di byte di dati. Il significato di questi è già stato descritto in dettaglio (vedere a pag. 1866), pertanto si può procedere alla realizzazione di un programma capace di ricevere dati MIDI e di elaborarli. Tuttavia il computer può essere usato anche per generare dati MIDI da trasmettere poi a un sintetizzatore, per attivarne i circuiti audio. Un esempio di questo secondo impiego si ha quando il computer viene adoperato come un sequenziatore, introducendo da tastiera sequenze musicali composte secondo la tecnica "step-time" (ossia programmando ciascun passo all'interno di una battuta). Tale operazione viene eseguita prima che la trasmissione attraverso la MIDI attivi il sintetizzatore per l'esecuzione della linea melodica. In realtà, questo lavoro è abbastanza semplice e il sistema MIDI può eseguire operazioni ben più complesse. Si pensi, per esempio, a un sequenziatore che adopera i 16 canali della MIDI per controllare una serie di tastiere e percussioni elettroniche!

Riuscendo a sviluppare un programma capace di registrare, in tempo reale, nella memoria del computer, una melodia suonata su un sintetizzatore, è possibile combinare alcune caratteristiche delle diverse modalità d'impiego del computer per interagire con gli strumenti collegati alla MIDI. Successivamente il programma, in risposta a opportune istruzioni, ritrasmette la melodia al sintetizzatore che la riproduce tramite la MIDI. Naturalmente la tecnica di registrazione è digitale.

Sebbene apparentemente simile a quella analogica, usata dai registratori a nastro magnetico, la registrazione digitale attraverso la MIDI si basa su principi assai diversi. Quando viene eseguita una registrazione su nastro, i suoni musicali vengono codificati sotto forma di configurazioni magnetiche sulla superficie del nastro: la loro posizione su di esso determina la sequenza di riproduzione dei suoni e le pause tra un suono e l'altro. Facciamo un esempio: due note registrate con questa tecnica e separate da una pausa di due secondi. Durante tale intervallo, il nastro continua a scorrere sotto le testine di registrazione anche se non viene memorizzato alcun suono. Quando il nastro viene "riletto", la lunghezza del segmento non registrato determina la pausa tra l'emissione della prima e della seconda nota.

In un sistema di registrazione digitale abbinato al MIDI, le due note corrispondono alla pressione di due tasti, ciascuno dei quali genera un messaggio MIDI. Una procedura di registrazione in tempo reale deve inoltre essere capace di registrare la pausa che intercorre tra la ricezione del primo e del secondo messaggio MIDI da parte del computer. I due messaggi, seguendo l'esempio della registrazione su nastro, potrebbero essere depositati in memoria adoperando una matrice e l'intervallo che li separa si potrebbe rappresentare scrivendo valori "zero" in una serie di byte, ciascuno corrispondente a una singola unità di tempo. Questa tecnica di registrazione in tempo reale dei messaggi MIDI consegue due



Registrazione digitale

Lo schema a fianco mostra come viene registrata una nota, usando il programma per la registrazione digitale. Durante la fase di registrazione, il sintetizzatore trasmette un messaggio «nota on» quando viene premuto un tasto e un messaggio «nota off» quando il tasto viene rilasciato. I messaggi vengono immagazzinati in memoria assieme all'informazione di pausa che indica l'intervallo di tempo, espresso in unità di 2 millisecondi, tra un evento e il successivo. Nell'esempio, l'intervallo tra i messaggi di nota on e off viene registrato nei due byte $F0 e $37. Durante la fase di riproduzione, il programma usa l'informazione di pausa per ritardare la trasmissione dei messaggi di nota e riprodurre il brano musicale con la massima "fedeltà"

importanti risultati: consente di riprodurre le due note nel momento stesso dell'emissione, conservando il corretto intervallo di tempo, e di risparmiare notevoli quantità di memoria.
Un metodo alternativo al precedente è quello di registrare l'informazione relativa alla pausa scrivendone il valore direttamente in un byte, senza inserire alcun byte nullo. Se l'intervallo di tempo tra le due note è, ad esempio, di 50 unità di tempo, il programma introduce nella matrice un byte di pausa contenente il valore 50 al posto di 50 byte "zero".
Se da un lato ciò consente un apprezzabilissimo risparmio di memoria, dall'altro rende più difficoltoso prevedere l'occupazione in memoria di una sequenza. Indicativamente, a parità di durata, i brani musicali caratterizzati da una rapida successione di note richiedono un maggior numero di messaggi "on-off" per le note (e pertanto più byte), che non brani con una minore concentrazione di variazioni tonali. Il frequente ricorso al modificatore di tonalità genera ulteriori messaggi MIDI che contribuiscono ad accrescere il consumo di memoria.

Programma di registrazione digitale

Il programma a fianco utilizza un particolare tipo di contatore (una variabile chiamata TEMP) per registrare gli intervalli di tempo tra due eventi successivi. La precisa misurazione delle pause e, di conseguenza, la "fedeltà" della registrazione dipendono dal ritmo col quale varia il contatore. La scelta della frequenza d'incremento ottimale è il risultato di un compromesso: nel programma qui presentato è stato fissato un intervallo di circa 2 millisecondi, non così piccolo da risultare minore del tempo d'esecuzione delle routine ma neppure così grande da influire sulla precisione della registrazione. Questo intervallo di tempo (detto "risoluzione" della registrazione, poiché rappresenta il più piccolo intervallo di tempo registrabile), viene generato da un temporizzatore situato in uno dei chip CIA o VIA del micro.
Il formato del messaggio MIDI deve essere leggermente ampliato per tener conto della temporizzazione quindi, prima di ogni messaggio MIDI, viene inserito un byte che rappresenta il numero di intervalli di tempo trascorsi dall'ultimo messaggio. Questo byte può registrare un massimo di 239 unità di tempo, ossia valori compresi tra $0 e $EF.
Se dopo 239 intervalli il computer non ha ricevuto un nuovo messaggio, viene registrato un errore di "overflow" (superamento della capacità) rappresentato dal byte $F0. Infine è stato scelto un messaggio a singolo byte, $FF, per segnalare la fine della sequenza. I valori $F1 e $FE non vengono utilizzati e sono a disposizione dell'utente.
Nella sezione conclusiva del progetto MIDI è presentata la versione per Micro BBC del programma di registrazione digitale e vengono forniti utili consigli anche per eventuali perfezionamenti del programma.



PROGRAMMA DI REGISTRAZIONE DIGITALE

Caricatore BASIC

```
10 FOR I=49152 TO 49581
20 READ X:POKE I,X:S=S+X:NEXT
30 READ X:IF S=X THEN PRINT"OK DIGITARE SYS 49152":END
40 PRINT "ERRORE DI CHECKSUM!"
50 END
100 DATA76,7,192,2,83,201,241,169,3
110 DATA141,0,222,169,21,141,0,222,169
120 DATA255,141,173,193,169,0,141,6
130 DATA220,169,8,141,7,220,169,17,141
140 DATA15,220,32,228,255,201,0,240
150 DATA249,201,69,208,1,96,201,82,240
160 DATA4,201,80,208,236,72,120,169
170 DATA127,141,0,220,169,173,133,247
180 DATA169,193,133,248,169,173,141,5
190 DATA192,169,241,141,6,192,160,0
200 DATA104,201,82,8,240,66,169,2,32
210 DATA69,193,32,44,193,201,255,208,4
220 DATA40,76,6,193,141,4,192,201,240
230 DATA173,4,192,240,12,32,246,192
240 DATA240,251,206,4,192,208,246,176
250 DATA223,32,44,193,72,173,0,222,41
260 DATA2,240,249,104,141,1,222,16,3
270 DATA32,53,193,202,208,233,174,3
280 DATA192,16,195,169,1,32,69,193,140
290 DATA4,192,162,255,32,246,192,240
300 DATA16,238,4,192,173,4,192,201,240
310 DATA144,6,32,21,193,140,4,192,173
320 DATA0,222,41,1,208,6,224,1,48,224
330 DATA16,243,173,1,222,48,11,224,0
340 DATA48,213,208,27,174,3,192,16,11
350 DATA201,248,176,223,201,240,176
360 DATA196,32,53,193,72,173,4,192,32
370 DATA21,193,140,4,192,104,32,21,193
380 DATA202,240,178,208,197,173,1,220
390 DATA73,239,208,18,104,104,40,208,4
400 DATA169,255,145,247,88,169,0,32,69
410 DATA193,76,37,192,173,13,220,41,2
420 DATA96,145,247,238,5,192,208,18
430 DATA238,6,192,208,13,104,104,104
440 DATA169,3,32,69,193,76,6,193,177
450 DATA247,230,247,208,2,230,248,96
460 DATA162,2,201,192,144,5,201,224
470 DATA176,1,202,142,3,192,232,96,10
480 DATA170,189,96,193,133,249,189,97
490 DATA193,133,250,160,0,177,249,240
500 DATA6,32,210,255,200,208,246,160,0
510 DATA96,104,193,134,193,146,193,158
520 DATA193,82,32,61,32,82,69,67,79,82
530 DATA68,44,80,32,61,32,80,76,65,89
540 DATA44,69,32,61,32,69,88,73,84,13
550 DATA0,147,82,69,67,79,82,68,73,78
560 DATA71,13,0,147,80,76,65,89,66,65
570 DATA67,75,32,13,0,79,85,84,32,79
580 DATA70,32,77,69,77,79,82,89,13,0
590 DATA240
600 DATA52178: REM *CHECKSUM*
```

Listato Assembly

```
*=$C000
IMMISS = $FFE4          ;ACCETTA UN CARATTERE
       OUTCAR   = $FFD2 ;VISUALIZZA UN CARATTERE
       JMP INIZIO
       NBYTE *=*+1      ;N. BYTE NEL MESSAGGIO CORRENTE
       TEMP *=*+1       ;PAUSA IN PERIODI DI 2 msec.
                0,3
       MEMLIB *=*+2     ;N. DI BYTE LIBERI
       DATAREG  = $DE01 ;REG. DI TRASM./RICEZ. SU ACIA
       STAREG   = $DE00 ;REG. DI STATO/CONTROLLO
       ARRESTO  = $EF
       MEM      = $F7   ;PUNTAT. ALLA MEMORIA DEI DATI
       PUNT     = $F9   ;PUNTAT. ALLA STRINGA LETTA
       START    = *
;+++++ PREPARA LE ROUTINE ++++
       LDA #$03
       STA STAREG       ;INIZIALIZZA IL CHIP 6850
       LDA #$16
```

MARCUS WILSON-SMITH

Funzionamento del programma

Il programma di registrazione digitale si basa su un algoritmo molto semplice. Quando viene selezionata l'opzione "registrazione", il programma registra i messaggi in arrivo presenti sulla porta MIDI IN. La registrazione si interrompe quando la memoria disponibile è piena oppure quando viene premuta la barra spaziatrice. L'opzione "riproduzione" ritrasmette la sequenza registrata attraverso la porta MIDI OUT. Se non viene premuta la barra spaziatrice, l'emissione prosegue fino al termine della sequenza. Un'eventuale nuova registrazione cancella quella precedente

```
      STA STAREG
      G10     LDA #$FF
STA   INIMEM          ;AZZERAMENTO INIZIALE
      LDA #$00
      STA $DC06
      LDA #$08
      STA $DC07
      LDA #$11
      STA $DC0F       ;AVVIA IL TEMPORIZZATORE B
      G20     JSR IMMISS
      CMP #$00
      BEQ G20
      CMP #69         ;PREMERE 'E' PER USCIRE
      BNE G22
      RTS
G22   = *             ;USCITA NON RICHIESTA
      CMP #B2
      BEQ G30
      CMP #B0
      BNE G20
                      ;++INIZIO PROG. IN TEMPO REALE++
      G30     PHA     ;SALVA COMANDO DI REGISTR/RIPROD
      SEI
      LDA #$7F
      STA $DC00       ;ESPLORA L'ULTIMA LINEA DI TASTI
      LDA #<INIMEM
      STA MEM
      LDA #>INIMEM
      STA MEM+1
      LDA #<BYTELIB
      STA MEMLIB
      LDA #>BYTELIB
STA   MEMLIB+1
      LDY #$00
      PLA
      CMP #B2
      PHP             ;SALVA IL FLAG Z=REGISTRAZIONE
      BEQ R00
                      ;
      P00 = *         ;ESEGUE LA ROUTINE
                      ;
      LDA #$02
      JSR STASTR
      P05 JSR READ
      CMP #$FF
      BNE P07         ;ESCE SE GIUNTO ALLA FINE
      PLP             ;AGGIUSTA LO STACK
      JMP C20         ;PROCEDURA DI USCITA
      P07 STA TEMP
      CMP #$F0        ;VALORE DEL CARRY=$F0 LEGGE
      LDA TEMP
      BEQ P30         ;SALTA SE NON DEVE ATTENDERE
      P10 JSR VERIF
      BEQ P10
      DEC TEMP
      BNE P10         ;SALTA SE NON E' SCADUTO TEMPO
      BCS P05         ;SE $F0, LEGGE PAUSA SUCCESSIVA
                      ;
      P30=*           ;LEGGE MESSAG. (BYTE) SUCCESSIVO
JSR   READ
PHA                   ;SALVA IL BYTE DI DATI
;
P35   = *             ;ATTENDE IN BASE AL REGISTRO TX
LDA   STAREG
AND   #$02
BEQ   P35             ;SALTA SE ANCORA PIENO
;
PLA                   ;ACCETTA DATI
STA   DATAREG         ;LI INVIA ALLA MIDI
BPL   P50             ;SALTA SE NON E' BYTE DI STATO
JSR   GETNO           ;ACCETTA NUMERO DEI BYTE DI DATI
P50   DEX             ;DECREMENTA IL CONTATORE DI BYTE
BNE   P30             ;RICEVE IL BYTE DI DATI SEGUENTE
LDX   NBYTE
BPL   P05             ;RICEVE IL BYTE DI CLOCK
;
R00   = *             ;ROUTINE DI REGISTRAZIONE
;
LDA   #$01
JSR   STASTR
STY   TEMP            ;TEMP = 0
R05   LDX #$FF        ;INIZIALIZZA A NESSUN DATO
R10   JSR VERIF
BEQ   R20
INC   TEMP
LDA   TEMP
CMP   #$F0
BCC   R20
JSR   MEMOR
STY   TEMP
;
R20   = *             ;CERCA DATI MIDI
LDA   STAREG
AND   #$01
BNE   R40             ;SALTA SE TROVA DEI DATI
CPX   #$01
BMI   R10             ;SALTA SE TROVA UNA PAUSA
BPL   R20             ;SALTA SE TROVA MESSAGGIO MIDI
;
R40   = *             ;ACCETTA IL DATO
LDA   DATAREG
BMI   R50             ;SALTA SE TROVA BYTE DI STATO
CPX   #$00
BMI   R10             ;SALTA SE MANCA INFORMAZ. CANALE
BNE   R80             ;SALTA SE TROVA MESSAGGIO MIDI
LDX   NBYTE           ;AZZERA IL CONTATORE DI BYTE
BPL   R60
;
R50   = *             ;ELABORA IL BYTE DI STATO
CMP   #$FB
BCS   R20             ;IGNORA SE SIST. IN TEMPO REALE
CMP   #$F0
BCS   R05             ;SALTA SE MANCA STATO DEL CANALE
JSR   GETNO           ;ACCETTA IL N. DEI BYTE DI DATI
;
R60   =*              ;REGISTRA IL DATO
PHA
LDA   TEMP
JSR   MEMOR
STY   TEMP            ;AZZERA IL CONTATORE
PLA
;
R80   = *             ;REGISTRA IL DATO MIDI
JSR   MEMOR
DEX
BEQ   R10
BNE   R20
;
VERIF = *
;
LDA   $DC01
EOR   #ARRESTO        ;LASCIA INALTERATO IL CARRY
BNE   C40
PLA
PLA                   ;AGGIUSTA LO STACK
PLP                   ;ESAMINA COMANDO REGIST/RIPROD
BNE   C20             ;SALTA SE REGISTRAZIONE
LDA   #$FF
STA   (MEM),Y
C20   = *
CLI
LDA   #$00
JSR   STASTR
JMP   G20             ;ATTENDE IL COMANDO SUCCESSIVO
;
C40   = *             ;ESAMINA IL TEMPORIZZATORE B
LDA   $DC0D
AND   #$02            ;FLAG Z = TEMPO NON SCADUTO
RTS
;
MEMOR = *             ;REGISTRA IL DATO IN MEMORIA
;
STA   (MEM),Y
INC   MEMLIB          ;CALCOLA LA MEMORIA LIBERA
BNE   AGGPUNT
INC   MEMLIB+1
BNE   AGGPUNT
PLA
PLA
PLA
LDA   #$03            ;MESSAGGIO D'ERRORE
JSR   STASTR
JMP   C20             ;IL PROGRAMMA RIPARTE DA CAPO
;
READ  = *             ;LEGGE UN DATO DALLA MEMORIA
;
LDA   (MEM),Y
;
AGGPUNT = *           ;AGGIORNA PUNTATORE ALLA MEMORIA
;
INC   MEM
BNE   P20
INC   MEM+1
P20   RTS
;
GETNO = *             ;RICEVE IL ACC. IL N. DEI BYTE
;                      DI DATI PER  LO STATO
;
LDX   #$02            ;INIZIALIZZA A 2 BYTE DI DATI
CMP   #$C0
BCC   N10
CMP   #$E0
BCS   N10
DEX                   ;1 BYTE DI DATI SE C0<=A<E0
N10   STX NBYTE
INX                   ;AGGIUNGE 1 PER LO STATO
RTS
;
STASTR = *            ;VISUALIZZA UNA STRINGA
;
ASL   A
TAX
LDA   TABMESS,X
STA   PUNT
LDA   TABMESS+1,X
STA   PUNT+1
LDY   #$00
M10   LDA (PUNT),Y
BEQ   M70
JSR   OUTCAR
INY
BNE   M10
M70   LDY #$00
RTS
;
;
TABMESS = *           ;TABELLA DEI MESSAGGI
;
 .WORD  MESS0
 .WORD  MESS1
 .WORD  MESS2
 .WORD  MESS3
;
MESS0   .BYTE 'R = REGISTRAZIONE, P =
                RIPRODUZIONE, E = USCITA', $0D,0

MESS1   .BYTE 147, 'REGISTRAZIONE', $0D,0

MESS2   .BYTE 147, 'RIPRODUZIONE', $0D,0

MESS3   .BYTE 'MEMORIA PIENA', $0D,0
INIMEM  = *           ;INIZIO MEMORIA PER DATI
BYTELIB = INIMEM - $D000
```

Il DrLOGO

Un confronto tra la versione per Amstrad e quella per il PC della IBM

Quando il LOGO è stato trasferito per la prima volta dai mainframe ai microcomputer sono stati necessari alcuni compromessi affinché il linguaggio potesse funzionare con i 64 Kbyte (o anche meno) di memoria a disposizione. Tuttavia, subito dopo il lancio dei micro a 16 bit, dotati di memorie molto più ampie, sono comparse sul mercato numerose versioni "avanzate" del LOGO.

Anche Gary Kildall, fondatore della Digital Research e creatore del CP/M, è rimasto affascinato dalle grandi potenzialità del LOGO tanto da sviluppare una propria versione del linguaggio, per il PC della IBM, chiamata DrLOGO. Oggi il DrLOGO è disponibile per altri computer a 16 bit, tra cui l'Apricot F1e (vedere a pag. 1869).

Inoltre la Digital Research ha prodotto versioni del linguaggio adatte a numerosi apparecchi CP/M a 8 bit (la Amstrad, ad esempio, lo ha inserito nel "package" di programmi applicativi per il CPC 464).

La versione per il PC IBM richiede uno spazio di memoria di almeno 192 Kbyte. Poiché il dischetto non può essere copiato, il pacchetto DrLOGO contiene una copia di riserva. Per caricare il linguaggio occorre inserire il dischetto nel drive ed eseguire il "reset" del sistema. La grafica a colori, con una risoluzione di 320x200 pixel, ha una tavolozza di 16 colori di sfondo e 12 di primo piano, suddivisi in 4 gruppi di 3 "penne".

Innestando sul PC una scheda per grafica a colori e collegando contemporaneamente al computer uno schermo a colori e uno monocromo, il DrLOGO può produrre grafica sullo schermo a colori e testi sullo schermo monocromo! Tuttavia, se si possiede un solo monitor, le due funzioni possono anche essere abbinate sullo stesso schermo.

Il manuale definisce il DrLOGO come un "insieme esteso" del LOGO Apple. In effetti il linguaggio, oltre ad offrire tutti i comandi LOGO per la grafica standard, l'elaborazione delle liste e la gestione delle aree di lavoro, possiede anche le "primitive" per la gestione degli errori, le "proprietà" (funzioni per l'assegnazione di più valori a un oggetto) e le procedure d'integrazione o package (istruzioni per integrare diverse procedure in un unico gruppo).

Si noti infine che, in DrLOGO, i comandi LOGO devono essere immessi usando le lettere minuscole.

Il programma di editor funziona tramite i tasti funzione IBM, ma riconosce anche i codici di controllo usati nel LOGO Apple. Se si presenta un errore durante l'esecuzione di un programma, il comando ed seleziona l'editor,

PROVE DI BANCO DEL LOGO

PROCEDURA		RISULTATI			
		IBM PC (256K)	AMSTRAD CPC 464/664	COMMODORE 64	APRICOT F1e
RECUR	Livello raggiunto	512	187	90	832
DEFVARS		1307	263	553	1606
ENDRECUR1		512	189	INFINITO	INFINITO
ENDRECUR2		151	129	65	72
LINES (tartaruga visibile)	Tempo (approssimato al secondo)	9	53	13	10
LINES (tartaruga invisibile)		5	39	11	6
ASSIGN		25	22	6	20
LISTS		64	74	18	69
ARITH		80	138	36	77
RECURA		92	91	24	72

LOGO A CONFRONTO

Le 9 procedure di prova "al banco" permettono di valutare le principali caratteristiche del LOGO e di scoprire i pregi e i difetti delle diverse versioni del linguaggio

- RECUR - misura le dimensioni dello stack
- DEFVARS - misura lo spazio disponibile per la definizione di variabili e procedure
- ENDRECUR1/ENDRECUR2 - valutano l'efficienza dell'elaborazione di procedure a ricursione finale. Una versione "ideale" dovrebbe consentire un numero infinito di ricursioni. Mentre ENDRECUR2 genera propri risultati e può essere considerata un'operazione, ENDRECUR1 è un comando
- LINES - misura la velocità di tracciamento. Le coppie di valori forniti dimostrano un fatto facile da intuire: il modo "tartaruga invisibile" consente una maggior velocità
- ASSIGN - misura la velocità di assegnazione dei valori alle variabili
- LISTS - misura la velocità degli operatori per l'elaborazione delle liste
- ARITH - misura la velocità delle funzioni aritmetiche
- RECURA - misura la velocità delle chiamate ricursive

che visualizza automaticamente la procedura errata consentendone la correzione. Tuttavia, osservando lo schermo, non è possibile sapere se ci si trova in ambiente editor. Molte altre versioni del LOGO visualizzano invece una speciale linea di editor sull'ultima riga dello schermo.

Nuove primitive

Oltre alle primitive usuali per l'elaborazione delle liste, il DrLOGO offre una serie di nuovi comandi:

sort: riordina una lista in sequenza alfabetica
shuffle: dispone gli elementi di una lista in ordine casuale
piece: permette di selezionare una parte di una lista.

Inoltre il DrLOGO possiede alcune funzioni di debugging:

watch: permette di osservare lo sviluppo di una procedura linea dopo linea
trace: visualizza i nomi delle variabili mentre vengono chiamate e mentre vengono definite
debug: suddivide lo schermo in due finestre. La finestra "debug" viene usata per visualizzare le informazioni fornite dai comandi trace e watch, mentre sulla finestra "program" compaiono i risultati del programma.

Le procedure all'interno dell'area di memoria si possono corredare con commenti usando il comando noformat; quando però occorre ulteriore spazio per l'esecuzione del programma, tali commenti possono essere eliminati. Infine il DrLOGO presenta un gruppo di nuove primitive per la gestione delle procedure:

follows: permette di definire l'ordine di visualizzazione delle procedure sullo schermo
potl: visualizza i nomi delle procedure che non vengono chiamate da altre procedure
poref <nome>: visualizza i nomi delle procedure che chiamano la procedura <nome>
pocall <nome>: visualizza i nomi delle procedure chiamate dalla procedura <nome>.

Queste funzioni supplementari permettono di configurare un ambiente di lavoro ottimale per la creazione di programmi LOGO.
Tuttavia il DrLOGO presenta alcuni limiti: non possiede procedure di collegamento con il codice macchina e per il trattamento dei file e, inoltre, il programma di editor manca della funzione di ricerca e sostituzione. Il manuale, comunque, è chiaro e ben articolato: contiene un'introduzione al LOGO di tipo didattico e un'esauriente sezione di consultazione, dedicata alle primitive.
Naturalmente molte funzioni supplementari del DrLOGO sono assenti nelle versioni per apparecchi a 8 bit, incidendo notevolmente sulla disponibilità di memoria e sulla velocità d'elaborazione dei programmi. Lo spazio di lavoro del LOGO viene misurato in "nodi": le versioni a 8 bit ne possiedono da 2.500 a 3.000 mentre il DrLOGO ne offre circa 10.000. Questa ampia riserva di memoria facilita la definizione delle procedure e l'elaborazione delle procedure ricursive. Nonostante ciò, il DrLOGO non riesce a gestire in modo efficiente gli algoritmi a "ricursione finale", ossia le procedure in cui la chiamata ricursiva si trova nell'ultima linea. Per questo motivo alcuni programmi, che su macchine Commodore o BBC vengono eseguiti all'infinito, in DrLOGO esauriscono lo spazio di lavoro disponibile dopo poche centinaia di chiamate ricursive. Per quanto riguarda la velocità, mentre la grafica del DrLOGO è sufficientemente rapida, l'elaborazione delle liste e delle funzioni aritmetiche è più lenta rispetto ad altre versioni a 8 bit del linguaggio.

La versione Amstrad

Il DrLOGO viene fornito assieme all'unità a dischetti dell'Amstrad ed è registrato sulla facciata opposta a quella del CP/M. Questa versione è stata modificata per utilizzare nel modo migliore le caratteristiche hardware dell'apparecchio. La grafica adotta lo stesso sistema a quattro penne del DrLOGO per il PC IBM ma i colori delle penne vengono selezionati introducendo una lista di valori che concorrono a formare il colore desiderato combinando in proporzioni diverse il rosso, il verde e il blu. Contrariamente alla versione IBM, che dispone di una gamma limitata d'istruzioni audio, la versione Amstrad presenta i comandi env (per l'inviluppo del volume), ent (per l'inviluppo del tono) e release che disimpegna i canali posti in stato di attesa da un comando di suono.
La versione Amstrad possiede molte funzioni previste dal LOGO Apple, tra cui le "proprietà" e la gestione degli errori, ma non le procedure d'integrazione (o package) né le primitive di debugging e le funzioni supplementari per la gestione delle procedure, previste dal DrLOGO per il PC IBM. Ciò è dovuto probabilmente alle limitate risorse di memoria dell'Amstrad. Infine manca la funzione define e, per cancellare un file, si è obbligati a uscire dal CP/M.
Il programma di editing è piuttosto lento ma possiede le stesse caratteristiche dell'editor DrLOGO per il PC IBM.
I comandi .examine e .deposit (PEEK e POKE), assenti sul PC IBM, consentono un limitato accesso al codice macchina. La documentazione di corredo è insufficiente e rimanda a un apposito manuale didattico e di consultazione, sebbene venga fornito un sintetico sommario di quasi tutti i comandi.
Il DrLOGO dell'Amstrad prevede uno spazio di lavoro composto da 2105 nodi, tipico dei sistemi a 8 bit. Come nella versione per il PC IBM, le procedure a ricursione finale vengono trattate in modo insoddisfacente. Per le funzioni grafiche, aritmetiche e di elaborazione delle liste, il DrLOGO dell'Amstrad è uno tra i LOGO a 8 bit meno veloci e la maggior parte delle operazioni sono tre volte più lente che sul Commodore 64.

Reti LAN e nuovi comandi BASIC

Ulteriori possibilità offerte dalle routine dell'Interface 1

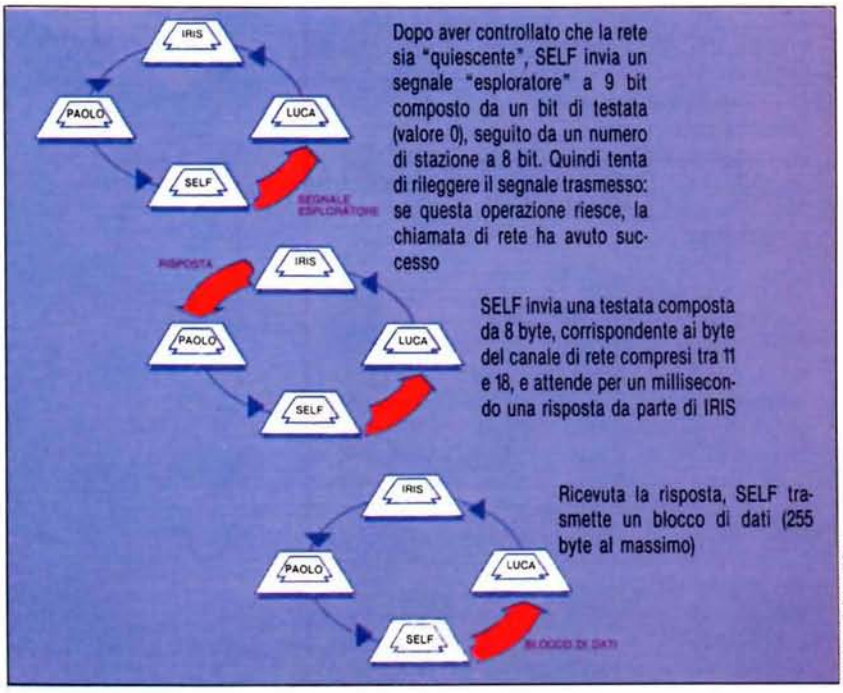
Si è già visto come il programmatore in codice macchina possa fruttuosamente impiegare le funzioni dell'Interface 1 per controllare i Microdrive e per accedere alla porta seriale RS232 (vedere a pag. 1876). Inoltre sono stati descritti i principi generali dei sistemi di trasmissione in rete (vedere a pag. 801) ed è stato anche presentato un semplice gioco "in rete" per lo Spectrum (vedere a pag. 826).
Qui vengono esaminate le funzioni di comunicazione in rete a disposizione del programmatore in codice macchina. Gli apparecchi collegati alla rete sono indicati con i seguenti nomi convenzionali: SELF per la macchina dell'utente e IRIS per quella con cui si desidera comunicare.
Esistono due metodi per trasmettere via rete programmi e dati. Il primo è la "trasmissione generica", che può essere ricevuta da qualsiasi apparecchio collegato alla rete, purché "sintonizzato" al momento dell'attivazione della comunicazione. Questa tecnica consente di inviare un programma o una serie di dati a più macchine contemporaneamente. Il secondo è la "trasmissione direzionata" nella quale SELF specifica quale IRIS debba ricevere l'informazione inviata. Se l'IRIS indicato non è in ascolto, SELF tenta ripetutamente di stabilire la comunicazione.
Le comunicazioni in rete adottano una procedura piuttosto interessante: l'apparecchio che intende trasmettere informazioni deve aspettare che la rete si trovi in stato di riposo o "quiescente"; per eliminare il rischio di accavallamenti durante la trasmissione, che potrebbero causare una commistione di dati, l'intervallo di attesa tra un tentativo e l'altro è diverso per ciascun computer collegato. Se la rete è "attiva", la macchina attende e riprova nuovamente.
Quando finalmente la rete risulta libera, l'apparecchio SELF che trasmette l'informazione invia un segnale "esploratore", che contiene il numero di stazione della macchina, configurato in modo da poter circolare nella rete ed essere riletto da SELF. In tal caso, SELF ha effettuato una "chiamata" della rete.
Viene quindi trasmesso un "pacchetto" di dati, composto da una testata (contenente informazioni sul blocco di dati seguente) e dal blocco di 255 byte di dati. Le operazioni successive dipendono dal tipo di trasmissione adottato. Se la trasmissione è direzionata, SELF attende per un millisecondo la risposta dell'IRIS chiamato. Se dopo 8 millisecondi non ha ricevuto alcun messaggio, invia nuovamente il segnale esploratore e la testata.
Il blocco di dati non viene trasmesso finché SELF non ottiene una risposta. Se la trasmissione è generica, SELF non attende alcuna risposta e invia immediatamente il blocco di dati sulla rete.
Esaminiamo adesso come la rete possa essere controllata dal codice macchina. Il procedimento coinvolge quattro codici di aggancio e due importanti variabili di sistema dell'Interfaccia 1:

- NTSTAT (locazione 23749): contiene il numero di stazione dell'apparecchio (in assenza, il valore 1). Viene creata dal comando FORMAT.
- DSTR1 (locazioni 23766 e 23767): questa variabile (usata anche quando sono collegati i Microdrive), contiene il numero di stazione dell'IRIS chiamato da SELF. Se il valore della variabile è zero, viene selezionata la trasmissione generica di dati.

L'impiego dei codici di aggancio assomiglia a quello necessario per i Microdrive, in quanto implica anch'esso l'apertura di un canale

Da stazione a stazione

Le comunicazioni in rete implicano, come prima operazione, una "chiamata" alla rete, seguita dalla trasmissione dei dati (preceduti naturalmente dalle necessarie informazioni di testata). Questo schema illustra il flusso dei dati su una rete a 4 stazioni dove la stazione SELF esegue una trasmissione "direzionata" verso la stazione IRIS



CAROLINE CLAYTON

(rappresentato dal diagramma a pagina seguente), che viene creato dal codice di aggancio 45. Il "canale di rete" ha una lunghezza di 276 byte e viene inserito, come qualsiasi altro canale, nell'area dei dati di canale della memoria.
La testata inviata alla rete è composta dai byte compresi tra NCIRIS e NCHCS incluso, mentre il blocco di dati corrisponde ai byte dello spazio di canale compresi tra 21 e 275. I byte 19 e 20 assumono significato solo quando il canale viene usato per leggere i dati in arrivo sulla rete. Come avviene con il canale dei Microdrive, la lettura e la scrittura vengono eseguite assegnando l'indirizzo del canale alla variabile CURCHL e usando le istruzioni RST #10 e CALL #15E6. Tuttavia, in questo caso, i dati e la testata vengono inviati alla rete quando si tenta di scrivere sul byte 256 dell'area dei dati di canale. Esaminiamo adesso i codici di aggancio:

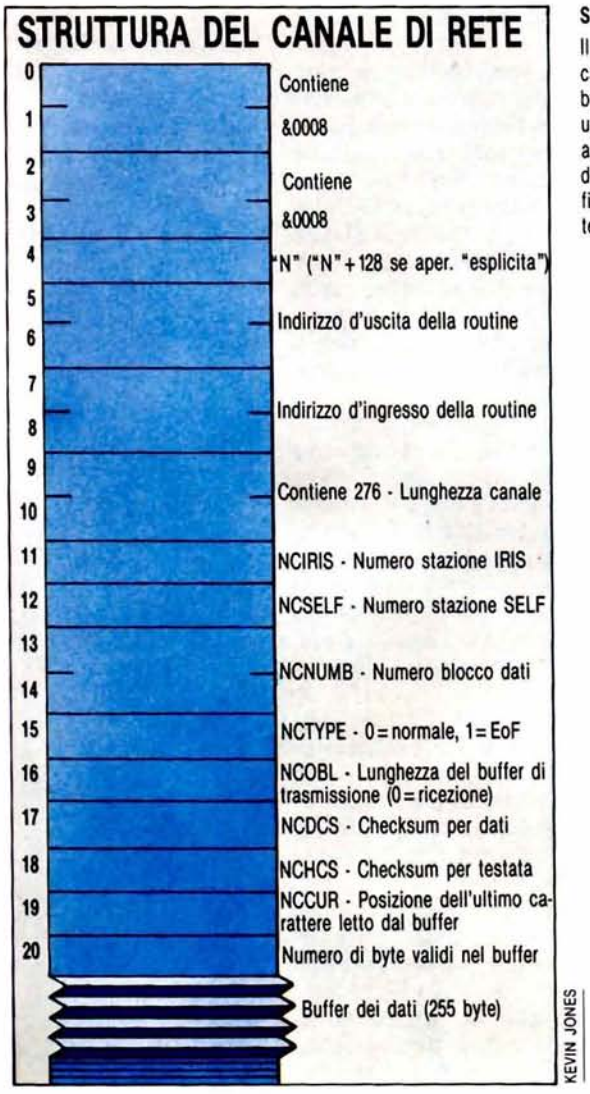
- *Codice di aggancio 45*: apre un canale di rete. Per usare questo canale è necessario assegnare a NTSTAT il numero di stazione di SELF e a DSTR1 quello di IRIS. Questa procedura permette di aprire il canale in lettura o in scrittura ma, nel secondo caso, è l'apparecchio IRIS che trasmette informazioni a SELF. Inoltre occorre assicurarsi che siano presenti le variabili di sistema dell'Interface 1. Come risultato, la routine del codice di aggancio 45 carica nel registro IX l'indirizzo iniziale del canale creato.
  Come già ricordato, i byte vengono trasmessi ricorrendo all'istruzione RST #10 e ricevuti tramite CALL #15E6. Tuttavia l'impiego di questi comandi può alterare il contenuto del registro IX, che contiene l'indirizzo iniziale del canale, utilizzato in seguito per chiudere il canale stesso, quindi è consigliabile memorizzare il contenuto del registro IX prima di usare queste routine.

- *Codice di aggancio 46*: chiude il canale di rete. L'indirizzo iniziale del canale da chiudere va depositato nel registro IX. Se il canale è stato usato per una scrittura, i dati che ancora si trovano nell'area dei dati di canale vengono inviati alla rete. Dopo che il canale è stato chiuso, l'area di memoria compresa tra la fine del canale e STKEND viene abbassata.

- *Codice di aggancio 47*: permette di leggere dalla rete uno specifico blocco di byte. Naturalmente la variabile di canale NCNUMB deve contenere il numero di blocco corrispondente e le variabili NCSELF e NCIRIS i numeri di stazione appropriati (che di solito vengono registrati al momento dell'apertura del canale). La routine tenta di ricevere il blocco indicato, ma se la ricezione non avviene nell'intervallo di tempo consentito oppure l'informazione risulta in qualche modo danneggiata e la "checksum" (somma di controllo) non corrisponde, il flag di carry viene posto a 1. In tal caso è necessario ripetere la richiesta del blocco desiderato. Se tutto funziona correttamente e il blocco di byte viene ricevuto, il sistema è in grado di inviare un segnale a conferma dell'avvenuta ricezione, dopodiché viene incrementato NCNUMB.

- *Codice di aggancio 48*: invia sulla rete un blocco di byte ed è, inoltre, il sistema principale per la trasmissione dei dati. Come sempre il registro IX deve contenere l'indirizzo iniziale del canale che, ovviamente, deve essere stato aperto in precedenza. Il registro A viene usato per indicare il tipo di file relativo al blocco di dati inviato dal canale. Se il registro è selezionato a 1, si tratta di un blocco "End of File" (blocco di fine file : E o F), altrimenti (registro A = 0) si tratta di un normale blocco dati.

La tecnica di trasmissione impiegata (generica o direzionata) dipende dal valore contenuto in DSTR1 al momento dell'apertura del canale. Quando i dati trasmessi vengono "accettati" dalla stazione ricevente, la testata viene registrata nell'area di testata del canale e il numero dei byte inviati depositato in NCOBL. La variabile NCTYPE indica se l'informazione ricevuta è un blocco EoF oppure no.
Se la trasmissione ha successo, la variabile di



Struttura del Canale di Rete
Il codice di aggancio 45 crea un canale di rete composto da 276 byte: i primi 20 byte vengono usati per la testata, mentre gli altri 255 sono un buffer per i dati da trasmettere. Lo schema a fianco descrive i byte dell'area di testata del canale

canale NCNUMB viene incrementata di 1.
La rete LAN permette di scegliere tra le due tecniche di trasmissione e ricezione dei dati sopra descritte. Tuttavia, se si opera su una rete di dimensioni ridotte collegabile solo ad un altro apparecchio, il metodo offerto dai codici di aggancio 48 e 47 appare leggermente sovradimensionato; conviene perciò applicare un sistema di comunicazione più semplice, basato sulle istruzioni RST #10 e CALL #15E6.

Nuovi comandi BASIC

L'Interface 1 permette di modificare le routine per la gestione degli errori dello Spectrum in modo da consentire l'accettazione di nuovi comandi BASIC. Per meglio comprendere queste operazioni, è bene esaminare le normali procedure d'individuazione degli errori dello Spectrum Sinclair.
Quando viene introdotta una nuova linea BASIC e viene premuto il tasto ENTER, l'interprete BASIC esamina immediatamente la linea, per controllare che non vi siano errori di sintassi, quali l'omissione dei nomi di variabile dopo i comandi NEXT, l'errato uso di parametri ecc., che ne renderebbero impossibile l'esecuzione. Se l'istruzione risulta corretta, il sistema la inserisce nel listato in memoria, se si tratta di una linea di programma, oppure la esegue se è un comando diretto.
Se però la linea contiene un errore, viene eseguita un'istruzione RST #8, seguita da un DEFB, che comunica al sistema operativo il tipo d'errore incontrato. Sullo schermo è visualizzato un punto interrogativo lampeggiante.
Quando la linea di programma o il comando diretto vengono elaborati, il sistema controlla nuovamente la sintassi valutando, ad esempio, le espressioni numeriche contenute nella linea e le esegue chiamando le opportune routine ROM. In questa fase vengono individuati gli errori relativi alle variabili (no such variable).
Mentre la prima verifica, detta *sintax check*, controlla che il comando sia sintatticamente corretto ma non lo esegue, la seconda, chiamata *run-time check*, lo invia in esecuzione. Tuttavia l'introduzione nel sistema dell'Interface 1 modifica questa procedura. Come già accennato, l'Interface 1 (IF1) consente di accedere in qualsiasi momento agli indirizzi #08 o #1708, di "impaginare" la ROM ombra e di escludere la ROM principale. Il sistema opera all'interno del nuovo "ambiente" (ROM ombra) finché non viene chiamato l'indirizzo #700 della ROM IF1 che disabilita la ROM ombra e ripristina la ROM principale.
Quando si verifica un errore, viene immediatamente abilitata la ROM ombra e il sistema esegue una seconda verifica sull'errore. Se l'errore è stato generato da un comando come CAT o FORMAT, considerati validi in presenza dell'Interface 1, l'esecuzione è passata all'appropriata routine della ROM della IF1, per poi tornare alla ROM principale.
Se invece la condizione d'errore dipende da altri comandi, il controllo passa alla variabile di sistema VECTOR della ROM IF1 (posta agli indirizzi 23735 e 23736), che contiene l'indirizzo della routine della ROM ombra per la gestione degli errori. Si noti che tale indirizzo varia a seconda della versione della ROM IF1 impiegata. Le principali differenze tra queste versioni vengono esaminate più avanti assieme ad una procedura per l'identificazione della ROM in uso.
Nella prima versione della ROM IF1, VECTOR contiene #1F0 e un eventuale errore genera una chiamata alla routine posta a tale indirizzo. Questa rinvia il programma di editor alla procedura *check error* e permette di correggere l'errore inserito nella linea BASIC sottoposta a verifica. Quindi, modificando l'indirizzo contenuto in VECTOR, il programmatore può reindirizzare la procedura per la gestione degli errori dello Spectrum in modo da eseguire una routine d'errore di propria progettazione sostitutiva di quella originale. Prima di descrivere nei dettagli questa operazione, conviene esaminare alcune routine della ROM ombra coinvolte nel processo.

#0010

Questa è l'unica routine che permette di chiamare una routine della ROM principale dall'interno dell'"ambiente ROM ombra". La procedura ha il seguente formato:

```
RST #0010
DEFW indirizzo; l'indirizzo della routine chiamata
```

#0020

Questa è la versione contenuta nella ROM ombra relativa alla routine RST #8 della ROM principale. L'istruzione #0020 è seguita da un singolo byte, che indica il messaggio d'errore da visualizzare. La seguente tabella evidenzia la gamma dei messaggi d'errore previsti:

Byte	Messaggio d'errore
00	Nonsense in BASIC - Privo di senso in BASIC
01	Invalid Stream Number - N. di flusso non valido
02	Invalid Device Expression - Indicazione di periferica non corretta
03	Invalid Name - Nome non valido
04	Invalid Drive Number - N. di drive non valido
05	Invalid Station Number - N. di stazione non valido
06	Missing Name - Nome mancante
07	Missing Station Number - N. di stazione mancante
08	Missing Drive Number - N. di drive mancante
09	Missing Baud Rate - Velocità di trasmissione (in baud) mancante
10	Header Mismatch Error - Errore d'accoppiamento tra Intestazioni
11	Stream Already Open - Flusso già aperto
12	Writing to a Read File - Tentativo di scrittura su un file in sola lettura
13	Reading a Write File - Tentativo di lettura di un file in scrittura
14	Drive Write Protected - Drive "protetto" in scrittura
15	Microdrive Full - Microdrive pieno
16	Microdrive Not Present - Microdrive non collegato
17	File Not Found - File non trovato
18	Hook Code Error - Errore nel codice di aggancio
19	CODE Error - Errore di codifica (CODE)

20 MERGE Error - Errore di fusione (MERGE)
21 Verification Has Failed - Verifica fallita
22 Wrong File Type - Tipo di file errato
255 Program Finished - Programma completato

#0028

Visualizza un messaggio d'errore "normale", ossia un messaggio generato dalla ROM principale e non dalla ROM ombra. Il codice d'errore corrispondente viene depositato nell'indirizzo 23610 prima che la routine venga chiamata.

#01F0

Genera il lampeggiamento del cursore di *syntax error*, che compare sullo schermo durante la verifica della sintassi.

#05B7

Questa routine, oltre a segnalare la conclusione della verifica sintattica del comando, individua la fine di un'istruzione BASIC (il funzionamento viene esaminato in seguito).

#05C1

Viene usata per concludere l'esecuzione di un comando nella ROM ombra e restituire il controllo alla ROM principale.

#0700

Questa routine ripristina l'ambiente controllato dalla ROM principale, disabilitando la ROM ombra a favore di quella principale.

Per concludere, esaminiamo il funzionamento del:

Codice di aggancio 50: come tutti gli altri, non andrebbe mai usato in "ambiente ROM ombra". Esso permette di chiamare le routine della ROM ombra dall'interno della ROM principale ma è consigliabile usarlo solo se si possiede una discreta conoscenza degli indirizzi della ROM ombra. Come mostra la seguente procedura, il codice 50 viene chiamato con l'indirizzo della routine della ROM ombra nella routine del registro HL:

```
LD      HL,ind: indirizzo della routine
LD      (23789),HL
RST     #8
DEFB    50
```

L'ambiente creato dalla ROM ombra presenta altre significative particolarità. Infatti, al suo interno, le istruzioni di restart dello Z80 hanno funzioni totalmente diverse. Inoltre non viene esaminata la tastiera né viene incrementata la variabile di sistema FRAMES. Infatti, il tempo impiegato per svolgere lavoro nella ROM non viene conteggiato in FRAMES, che risulta quindi inaffidabile nelle applicazioni in cui il tempo è un fattore critico.
Un ulteriore problema riguarda l'impiego della "calcolatrice in virgola mobile" (FPC) in "ambiente ROM ombra".
Poiché la routine all'indirizzo #0010 non sembra in grado di eseguire la chiamata per la FPC, conviene disabilitare la ROM ombra e ripristinare la ROM principale prima di selezionare questa funzione. Per riabilitare la ROM ombra si può usare il codice di collegamento 50.
Fin qui, la teoria: nell'ultima parte per il codice macchina dello Spectrum viene esaminato l'aspetto pratico dell'aggiunta di nuovi comandi al BASIC.

CURIOSARE NELLA ROM OMBRA

Disassemblare un programma scritto da altri è sempre istruttivo. Nel caso delle routine della ROM ombra, ciò facilita la comprensione del loro funzionamento e permette di utilizzarle in seguito con maggior efficienza. Sfortunatamente non è possibile esaminare (PEEK) la ROM ombra direttamente dal BASIC, poiché essa viene "impaginata" solamente durante il funzionamento dell'Interface 1. Questo programma BASIC permette di caricare una parte della ROM ombra in RAM, dove può essere esaminata

```
 10 GO SUB 200
 20 INPUT "Quanti byte vuoi riservare (CLEAR)?";b
 30 CLEAR r-(b+24)
 40 GO SUB 200
 50 FOR n=1 TO 23
 60 READ d: POKE r+n,d
 70 NEXT n
 80 INPUT "Indirizzo di partenza nella ROM Ombra ?";s
 90 INPUT "Numero di byte da copiare ?";b
100 LET z=s: GO SUB 300
110 POKE r+12,l: POKE r+13, h
120 LET z=b: GO SUB 300
130 POKE r+18,l:POKE r+19,h
140 LET z=r+9: GO SUB 300
150 POKE r+2,l: POKE r+3,h
160 LET z=r+24: GO SUB 300
170 POKE r+15,l: POKE r+16,h
180 RANDOMIZE USR (r+1)
190 PRINT "Indirizzo iniziale dei dati memorizzati"; r+24
195 STOP
200 LET r=PEEK 23730+256*PEEK 23731: RETURN
300 LET: h=INT(z/256): LET l=z-256*h: RETURN
400 DATA33,0,0,34,237,92,207,50,255,225,33,0,0,17,0,0,1,0,0,
    237,176,199,201
```



UNO SGUARDO... NELL'"OMBRA"

Oltre alle funzioni per il controllo dei Microdrive, della porta RS232 e dei sistemi in rete offerte dall'Interface 1, la ROM ombra possiede numerose routine di grande utilità, tra cui un convertitore da decimale a esadecimale (non previsto da alcuna delle altre subroutine di sistema). In merito esistono numerose pubblicazioni specializzate che contengono suggerimenti sul modo migliore per sfruttare l'interfaccia tramite il codice macchina. Il volume mostrato qui a fianco contiene una dettagliata descrizione delle routine della ROM ombra e alcuni interessanti programmi di utilità

Tra il dire e il fare...

In teoria i robot hanno infinite applicazioni, ma in pratica i limiti sono molti



ROBYN BEECHE

Una "vedette" cibernetica

Esistono robot costruiti anche per finalità molto curiose. La foto mostra "Rosa Bosom" (il nome deriva da: Radio Operated Simulated Actress Battery Or Stand-by Operated Mains), assieme al suo inventore Bruce Lacey. Realizzato nel 1966, questo robot è stato impiegato per il ruolo della Regina di Francia in una rappresentazione teatrale dei "Tre moschettieri", fatta dal Royal Court Theatre di Londra. Fabbricato con relais e motori di recupero, Rosa accettava comandi espressi sotto forma di note musicali, rispondendo ad essi con particolari movimenti. Inoltre, alcuni sensori ultrasonici consentivano al robot di evitare ostacoli durante gli spostamenti. La "parte" di Rosa era registrata su nastro e, tramite comandi a distanza, le "labbra" venivano mosse per mimare la recitazione. Da allora, Rosa è apparsa in diverse manifestazioni e mostre, vincendo addirittura il titolo di "Miss Mondo" in una competizione per esponenti... alternative del gentil sesso

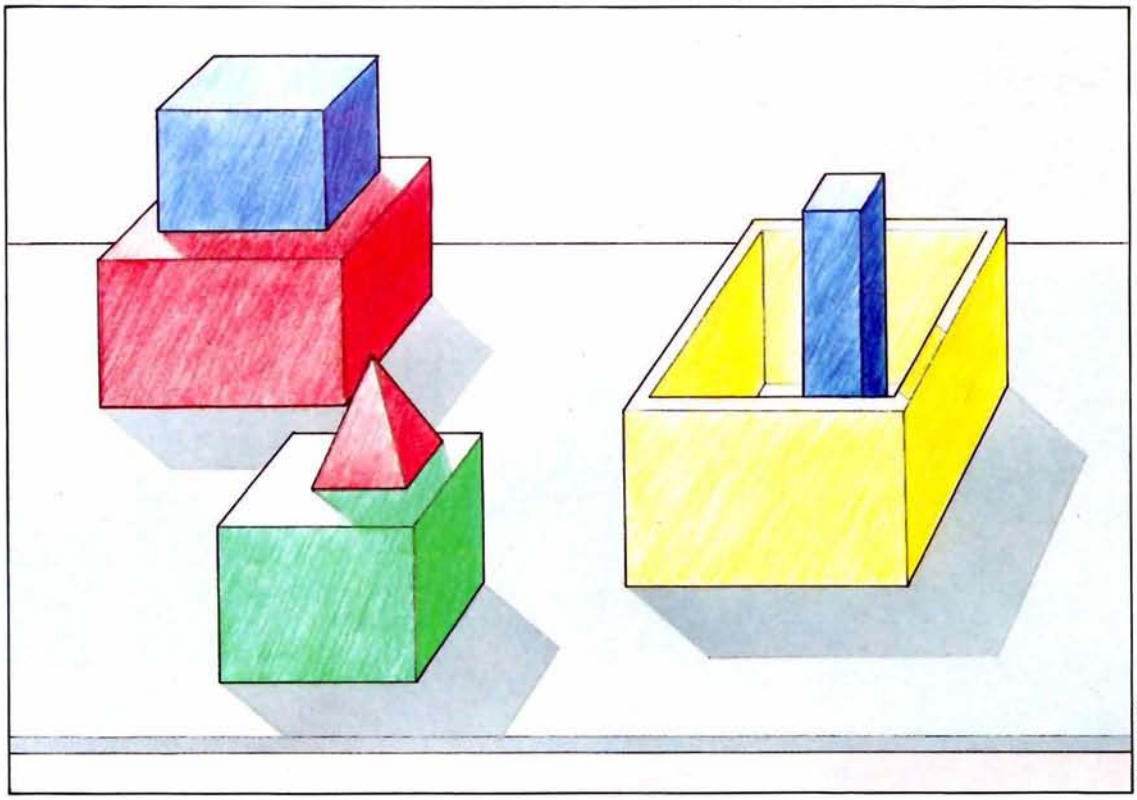
In nessun altro settore dell'informatica esiste un limite così netto tra fantasia e realtà, come in quello della robotica. Nei film hollywoodiani i robot sono capaci di camminare, parlare e perfino complottare contro l'uomo. Non lontano da Hollywood (alla Stanford University, in California), i più recenti prodotti della robotica offrono uno spettacolo ben diverso: sono a malapena capaci di muoversi per una stanza senza travolgere quello che incontrano sul proprio cammino.

I robot utilizzati nelle grandi industrie, particolarmente quelle automobilistiche, sembrano apparecchi dotati di maggiore destrezza, ma in realtà non fanno che ripetere innumerevoli volte una sequenza programmata di azioni, quali la verniciatura o la saldatura di pezzi. Se, per caso, il nastro trasportatore che convoglia i pezzi subisce un guasto, il robot continua imperterrito a verniciare... il vuoto!

Le ragioni di una tale discrepanza tra teoria

e realtà sono molto complesse, ma possono essere riassunte in due parole: "percezione" e "pianificazione". In primo luogo, gli attuali robot, seppur dotati di perfezionati sensori, non sono completamente capaci di interpretare i segnali rilevati. In secondo luogo, anche ammettendo un certo grado di interpretazione, i robot non sanno poi come utilizzare i dati raccolti. Ecco perché la robotica è un fertilissimo campo di ricerca per l'Intelligenza Artificiale che, in sostanza, studia il modo di imitare il comportamento umano ricorrendo a dispositivi elettronici e meccanici.

nel mondo reale giustificano l'enorme successo dei "Micromouse contest", ossia delle competizioni in cui un minuscolo veicolo computerizzato (chiamato "mouse") deve trovare da solo la via d'uscita da un labirinto. Infatti, nonostante i metodi e le teorie necessarie per raggiungere lo scopo siano ormai noti da anni, il piccolo "topo" deve confrontarsi con aspetti molto concreti quali la non perfetta linearità delle pareti del labirinto o magari la presenza di piccole macchie d'olio sul percorso, che fanno slittare i meccanismi. Per queste applicazioni robustezza e adattabilità



KEVIN JONES

Mondo dei blocchi o dei balocchi?

I cosiddetti "Mondi dei blocchi" sono ambienti altamente schematizzati in cui vengono impiegate forme geometriche abbastanza elementari. Tali mondi sono adoperati dai ricercatori di Intelligenza Artificiale per sperimentare metodi di analisi ambientale, di comprensione di testi e di risoluzione di problemi. Tuttavia, queste rimangono pur sempre delle semplificazioni del mondo reale, utili spotrattutto per la teoria

Tecnicamente non vi sono ostacoli insormontabili nel dotare un robot di tutta una vasta serie di sensori (telecamere, termostati, scansori ultrasonici, rilevatori di pressione e via dicendo), alcuni dei quali capaci di misurazioni inimmaginabili per l'uomo, i cui sensi hanno limiti piuttosto precisi. Tuttavia, se il robot non è immerso in un ambiente operativo ben definito ed altamente controllato, vi sono scarse probabilità che riesca a "comprendere" quanto i sensori inviano al suo "cervello". Il processo percettivo dell'uomo invece, come si è già visto in precedenza, è influenzato da modelli psicologici basati sull'esperienza e su infiniti fattori del mondo reale.

Un conto è formulare un algoritmo capace di far muovere razionalmente un piccolo veicolo computerizzato all'interno di un labirinto, un altro è rendere operativo tale algoritmo per pilotare un robot in un ambiente reale, ad esempio, quello urbano, in cui si incontrano imprevisti di ogni tipo (marciapiedi sconnessi, lattine accartocciate e via dicendo).

In un certo senso, queste difficoltà presenti sono più importanti dell'eleganza di un sofisticato algoritmo. In quasi tutti i campi dell'Intelligenza Artificiale, lo sperimentatore può isolarsi in "micromondi" che permettono almeno in larga misura di ignorare i dettagli della realtà. I sistemi esperti, tuttavia, devono lavorare su "fatti" e non su "cose", ecco perché i ricercatori di AI hanno una così alta considerazione dei cosiddetti "Mondi dei blocchi".

I "Mondi dei blocchi"

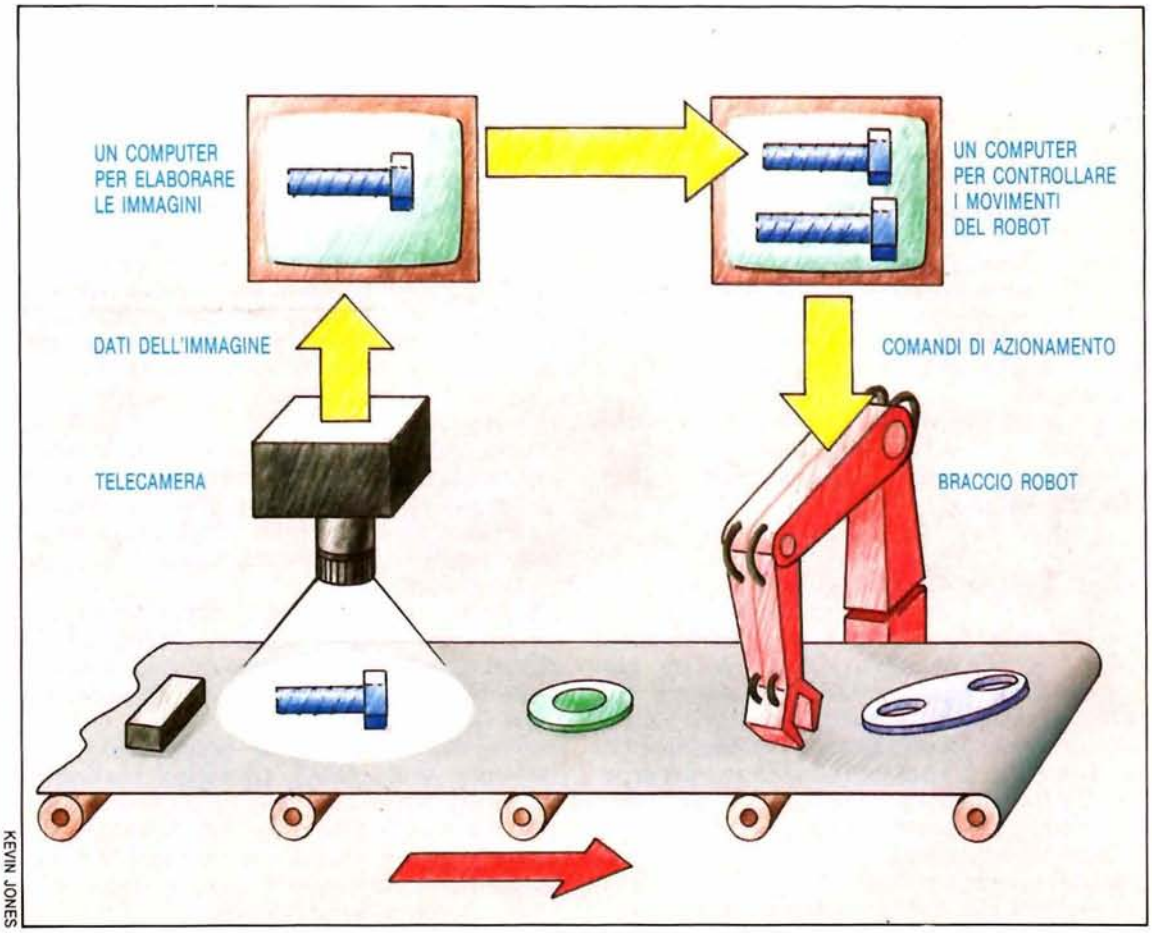
Quasi a sottolineare il parallelismo tra i processi percettivi dei bambini e quelli delle macchine, gli studiosi di Intelligenza Artificiale amano condurre esperimenti in ambienti semplificati, nei quali vengono impiegati blocchi per costruzioni simili a quelli con i quali giocano i bambini. Ciò permette di schematizzare la manipolazione di oggetti aventi varie forme.

Al contrario, la robotica non ammette una simile comodità: i robot devono confrontarsi

con la realtà e non con semplici giocattoli! Un robot "intelligente" deve essere quindi capace di crearsi un modello dell'ambiente circostante e, per rispettare le più antiche tradizioni, dovrebbe essere in grado di riconoscere ed eseguire comandi verbali. Per di più, poiché non tutte le situazioni sono prevedibili, dovrebbe anche possedere una buona capacità di apprendere dall'esperienza. Per raggiungere tali scopi, occorre prima risolvere almeno i principali problemi legati all'Intelligenza Artificiale:

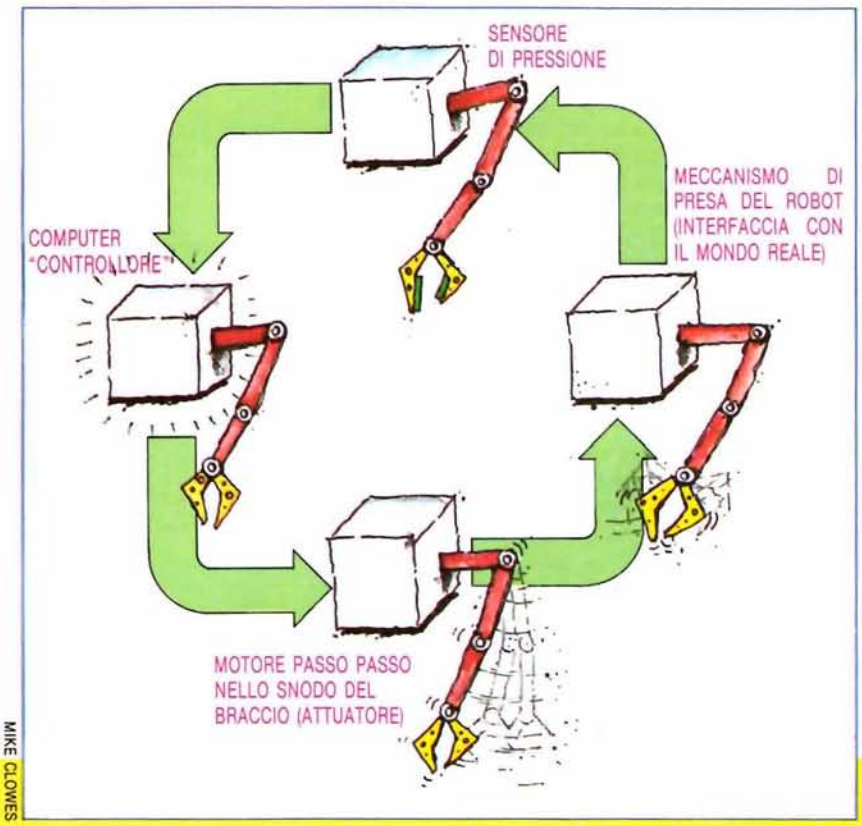
- La percezione tramite computer
- Il riconoscimento della lingua parlata
- La soluzione autonoma di problemi in un ambiente dinamico
- L'apprendimento da parte delle macchine.

I robot utilizzati nel settore industriale sono apparecchi estremamente finalizzati per l'impiego in ambienti altamente strutturati: di solito un dispositivo simile a un braccio, controllato da computer e operante in un preciso posto di lavoro. Sulla sua "mano" possono essere fissati svariati utensili e un operatore umano provvede ad "insegnare" i corretti movimenti durante varie sessioni di tirocinio. Molto spesso non occorre nemmeno che il robot sia dotato di capacità percettive.
Per altre applicazioni, però, occorrono robot più sofisticati e capaci di avere contatti con il mondo circostante; il metodo più comune è quello di abbinare al robot una telecamera controllata, spesso, da un computer diverso da quello che pilota il robot. In tal caso il robot diventa capace di afferrare e di lavorare oggetti che corrispondono (anche se in modo non estremamente preciso) alla forma schematica memorizzata. In sostanza, uno dei due computer riceve ed elabora l'immagine, l'altro decide quale azione far intraprendere al robot nel caso specifico.
Un vantaggio offerto anche da un elementare sistema visivo è che l'angolazione con la quale i pezzi da lavorare vengono presentati al robot non compromette i risultati: il computer, riconosciuto il pezzo e la sua angolazione, fa sì che il robot maneggi l'oggetto nel modo corretto.
Un secondo criterio di percezione è affidato ai sensori tattili, fissati in posizioni critiche, quali le estremità sporgenti o le "mani" del robot. Quando la mano "sente" di aver afferrato un oggetto, invia tale informazione al computer che provvede a passare all'azione successiva.
Sensori simili possono essere utilizzati anche nel riconoscimento dell'ambiente circostante: se il braccio urta un ostacolo, ad esempio, il computer ne memorizza la posizione e provvede ad evitarla in futuro. L'estrema accuratezza di questi sensori tattili viene spesso dimostrata con una semplice esperienza: in un primo momento al robot viene fatto trapanare un solido blocco di metallo, poi a questo viene sostituito un uovo: il robot trapassa il guscio senza rompere l'uovo.



KEVIN JONES

Afferrare il concetto e l'oggetto!
Un robot "vedente" può essere adoperato per riconoscere e raccogliere particolari oggetti da un nastro trasportatore. Spesso vengono utilizzati due computer: uno per l'analisi dell'immagine captata, l'altro per controllare i movimenti del robot



MIKE GLOWES

CICLO DI FEEDBACK

Per manipolare con precisione vari oggetti è necessario dotare il robot di una capacità tattile. Quando un uomo raccoglie un oggetto, automaticamente ne valuta il peso, per decidere quanta forza occorre applicare per sollevarlo, e la consistenza, per evitare di romperlo durante la presa. Acquisire simili risultati con un robot implica il ricorso ad accurati sistemi di feedback, nei quali le letture dei sensori di pressione servono per far decidere al computer quanta forza applicare

Sia la "vista" che il "tatto", nonostante siano ancora ad un livello primitivo nei robot, confermano l'importanza del "feedback", ossia di un metodo per analizzare i risultati di particolari azioni.
Trasformare i dati provenienti dall'esterno in una forma comprensibile al computer non è cosa facile, specialmente se i risultati devono essere in "tempo reale", ossia tanto veloci da poter reagire ad un evento prima che esso si esaurisca.
Quei pochi robot dotati di sensori e di un qualche sistema per l'interpretazione del feedback appartengono alla cosiddetta "seconda generazione".

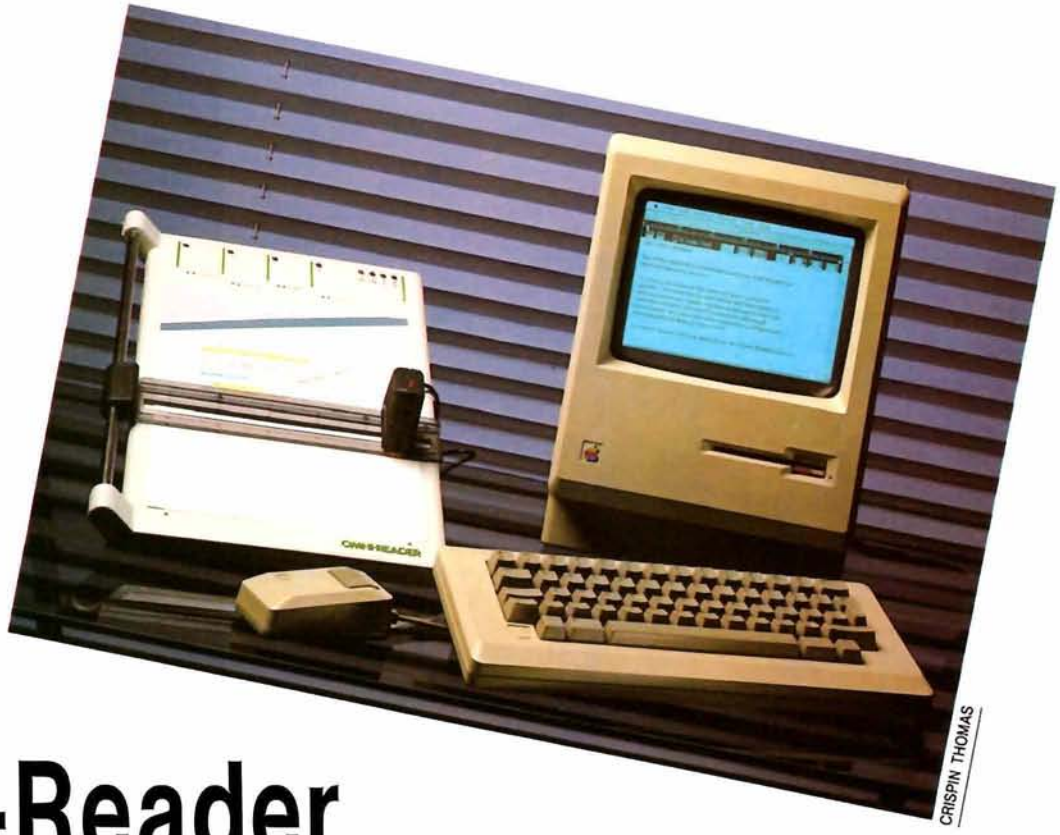
Una parola di comando

Le difficoltà da superare, affinché un robot riesca a comprendere intere frasi di una particolare lingua parlata, sono enormi. Tuttavia, è possibile insegnare ai robot il riconoscimento di alcune parole, o combinazioni di esse, purché formulate secondo precisi schemi. Sebbene ancora a livello molto elementare, questo tipo di interazione fra uomo e macchina risulta molto utile in ambienti di lavoro dove l'operatore, magari con le mani impegnate con qualche altro apparecchio, si trovi nella necessità di comunicare con il robot.
Ovviamente ciò non significa né che il robot "comprenda" né che "interpreti" il comando: semplicemente lo "riconosce" tra quelli memorizzati in precedenza nei suoi circuiti.
Un settore che ha prodotto interessanti sviluppi nel campo dei robot intelligenti è quello spaziale. La NASA ha progettato un veicolo computerizzato semovente, da utilizzare nelle esplorazioni superficiali del pianeta Marte, capace entro certi limiti di prendere le proprie decisioni in base a quanto rilevato dai sensori di cui è dotato. In simili circostanze, infatti, è impensabile attendere che i segnali emessi dai sensori vengano inviati sulla Terra e che i conseguenti comandi vengano poi trasmessi al robot: anche le più elementari operazioni richiederebbero almeno mezz'ora di tempo. La realizzazione di questo veicolo robotizzato è ancora lontana, ma già nel 1976 sono atterrate su Marte due sonde Viking, capaci di analizzare campioni del suolo e dell'atmosfera marziana senza un continuo e diretto controllo da Terra.
Ci sono enormi vantaggi nell'impiego di robot al posto degli uomini nei progetti spaziali: non ultimo quello di spendere molto meno! È stato calcolato che il costo orario per l'invio, la permanenza e il ritorno di un uomo dallo spazio sia valutabile intorno ai 10.000 dollari. I robot costano molto in fase di progetto e di realizzazione, ma poi possono essere utilizzati con poca spesa.

Un sistema che si autoriproduce

Negli anni '40 John von Neumann, il celebre matematico che ha gettato le basi teoriche per la realizzazione dei moderni computer, ha formulato un'esotica ipotesi per il futuro: automi in grado di riprodursi da soli.
Secondo von Neumann, le regole di tale riproduzione sono abbastanza elementari e presuppongono quattro elementi. Il primo è l'esistenza di una fabbrica automatizzata, in grado di estrarre o raccogliere le materie prime e di lavorarle secondo precise istruzioni. Il secondo componente è un "duplicatore" che riproduca tali istruzioni. Il terzo è un "controllore", che passi le istruzioni al duplicatore. Il quarto ed ultimo elemento è lo stesso insieme di istruzioni, che comunica al sistema come costruire una fabbrica automatica completamente nuova, utilizzando i prodotti ottenuti.
Queste considerazioni sono rimaste pura teoria per oltre 40 anni, finché i ricercatori della NASA non hanno proposto la realizzazione di un sistema autoriproducente, da collocare sulla Luna.
Questa fabbrica spaziale, dotata di un "costrotture universale", dovrebbe realizzare nuove fabbriche seguendo un processo molto simile a quello ideato da von Neumann. Le materie prime verrebbero estratte direttamente sulla Luna e non sarebbe necessario alcun intervento dalla Terra.

Adesso leggi tu!
L'Omni-Reader della Oberon è uno fra i primi dispositivi per il riconoscimento ottico di caratteri (OCR) ad essere al tempo stesso affidabile e relativamente poco costoso. Il testo stampato o scritto a macchina (una lettera, ad esempio) viene letto e automaticamente inviato, tramite porta seriale RS232, al computer collegato, che provvede a visualizzarlo e a memorizzarlo. Una volta in memoria, il testo può essere elaborato come di consueto con un word-processor oppure, se la lettura riguarda cifre e numeri, anche da un programma di contabilità



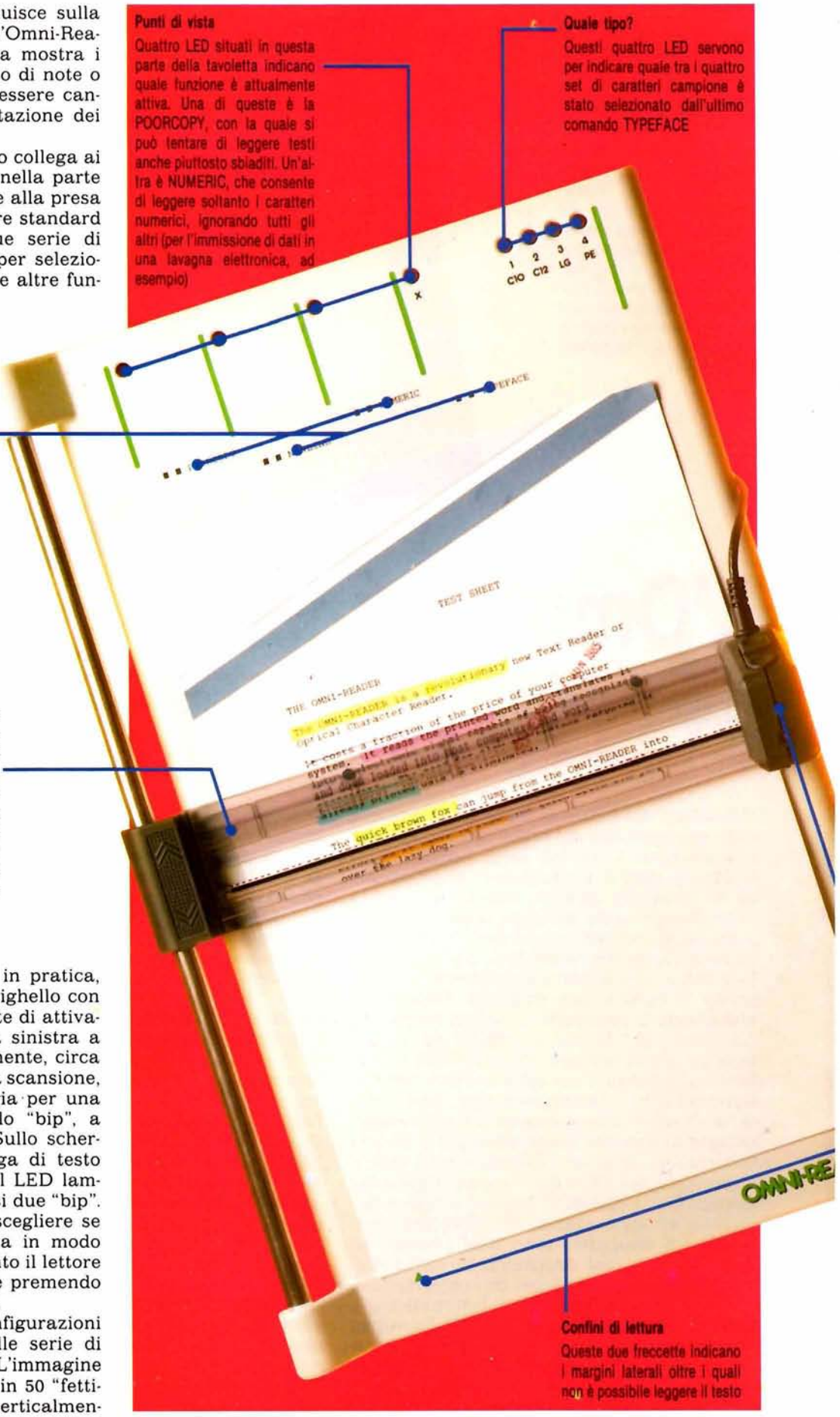
CRISPIN THOMAS

L'Omni-Reader

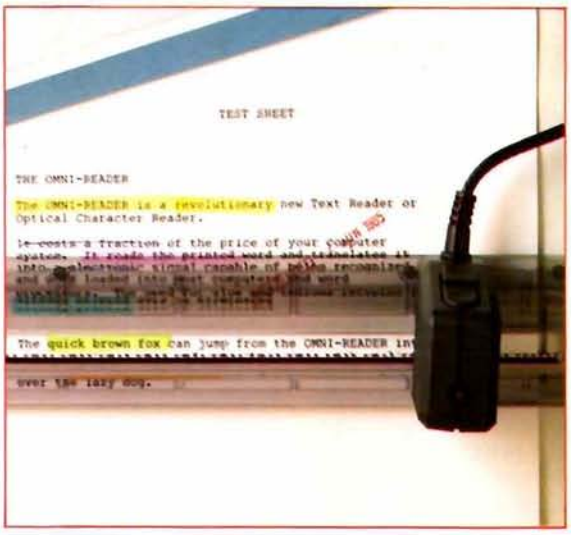
Un dispositivo che legge i testi e li invia al computer tramite una porta RS232

Nell'ufficio del futuro la maggior parte della sostanza cartacea dovrebbe sparire. Questa affermazione è basata sulla possibilità di trasferire dati e informazioni direttamente da un computer all'altro, eliminando così la necessità di usare la carta come supporto. Tuttavia, questo obiettivo è ancora abbastanza lontano, poiché l'abitudine, tra l'altro molto pratica, di annotare appunti con carta e penna è tuttora ben radicata. Comunque, nonostante il passaggio a questa nuova impostazione del lavoro in ufficio sia ancora lontano, esiste sempre un'operazione che finora ha implicato l'uso della tastiera: l'immissione dei dati. A facilitare le cose, molte ditte di hardware si sono specializzate nella realizzazione di appositi lettori ottici per il riconoscimento di caratteri. Recentemente, questi apparecchi per OCR (da "Optical Character Recognition"), capaci di leggere e convertire un testo scritto in segnali elettrici comprensibili da un computer, sono stati prodotti su larga scala e resi disponibili (e sufficientemente affidabili) anche per uso commerciale. L'Omni-Reader della Oberon International è uno dei più recenti dispositivi OCR basati su microprocessore, adatti anche ai personal e agli home computer, purché dotati di un'interfaccia seriale RS232.
L'Omni-Reader è composto da una tavoletta porta-documenti, dotata di una specie di righello orizzontale scorrevole in senso verticale. Il lettore ottico, fornito di due pulsanti e di un LED, è montato sul righello e può pertanto scorrere su tutta la superficie del documento da "leggere".
Esistono vari modi di funzionamento, che includono anche la scelta tra diversi tipi di carattere, indicati dall'accensione di una serie di LED posti sulla parte superiore della tavoletta. L'Omni-Reader, per il momento, è in grado di riconoscere soltanto 4 diversi tipi di carattere: il Courier 10, il Courier 12, il Lettera Gotica 12 e il Prestige Elite 12, corrispondenti a quelli impiegati sulla maggior parte delle stampanti a margherita e macchine da scrivere elettroniche.
All'interno del lettore ottico si trovano due sorgenti a raggi infrarossi, poste ai fianchi di una lente che focalizza l'immagine sul captatore di luce. Quest'ultimo è capace di "leggere" soltanto caratteri scritti con inchiostri a base di carbone o riprodotti con processo di fotocopiatura. Questa tecnica ha i suoi pro e i suoi contro. Un vantaggio è, ad esempio, che eventuali scritte fatte con inchiostri non a base di carbone risultano invisibili al lettore, pertanto il testo stampato risulta leggibile nonostante la presenza di scritte o di timbrature che coprano parzialmente il testo. Inol-

tre, il colore della carta non influisce sulla qualità della lettura da parte dell'Omni-Reader. Al contrario, però, il sistema mostra i propri limiti se il testo è costellato di note o segni fatti a matita, che devono essere cancellati per una corretta interpretazione dei caratteri da leggere.
Al lettore è fissato il cavetto che lo collega ai circuiti dell'Omni-Reader, situati nella parte posteriore della tavoletta, assieme alla presa per l'alimentazione e al connettore standard RS232 a 25 vie e, infine, a due serie di microinterruttori DIP (necessari per selezionare la velocità di trasferimento e altre funzioni).



Per la lettura di un documento, in pratica, basta allineare la finestrella del righello con la riga di testo, premere il pulsante di attivazione e far scorrere il lettore da sinistra a destra. Se tutto procede regolarmente, circa un secondo dopo aver effettuato la scansione, il LED posto sul lettore lampeggia per una volta e viene emesso un singolo "bip", a conferma dell'avvenuta lettura. Sullo schermo del computer compare la riga di testo appena letta. In caso di errore, il LED lampeggia due volte e vengono emessi due "bip". A questo punto l'operatore può scegliere se ripercorrere la riga di testo letta in modo parziale oppure se accettare quanto il lettore è riuscito a decifrare (ciò avviene premendo un apposito pulsante sul lettore).
L'Omni-Reader confronta le configurazioni estratte dal lettore con una delle serie di forme di caratteri a disposizione. L'immagine percepita dal lettore è suddivisa in 50 "fettine", prima orizzontalmente poi verticalmen-



Una finestra per leggere

Per un corretto funzionamento del dispositivo di lettura, la riga di testo deve trovarsi completamente all'interno della finestrella di lettura ricavata sul righello. Si noti, nella parte inferiore della finestra, una serie di trattini scuri: essa serve per indicare al software la direzione del movimento impresso al lettore ottico

OMNI-READER

INTERFACCE

L'interfacciamento con il computer avviene tramite una porta seriale standard RS232, con possibilità di scelta della velocità di trasmissione

PRO

Il funzionamento dell'Omni-Reader è rapido ed efficiente. Con un po' di pratica la velocità di trascrizione è pari a quella di un dattilografo

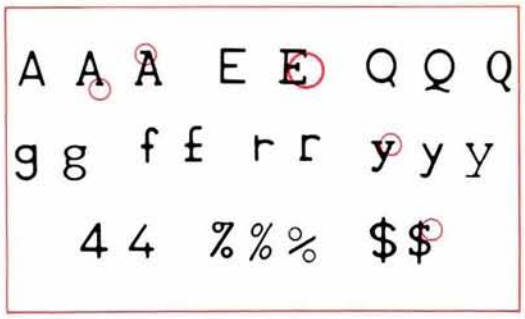
CONTRO

Esistono alcune limitazioni all'applicazione dell'Omni-Reader. Per adesso i set di caratteri riconosciuti perfettamente sono soltanto quattro (anche se scelti tra i più comuni). Un'altra limitazione è dovuta al fatto che il captatore ottico riesce a leggere soltanto caratteri stampati con inchiostri a base di carbone



Ottica pilotata

Il captatore ottico viene fatto scorrere a mano lungo il righello. Giunto in fondo a una riga, il singolo lampeggio di un LED e l'emissione di un "bip" confermano l'avvenuta lettura. Se il riconoscimento è parziale, vengono emessi due "bip", ma se il LED rimane acceso e l'emissione è continua, l'intera riga di testo è risultata incomprensibile



Caratteri incerti

Benché a prima vista i caratteri di stampa sembrano spesso simili, esistono caratteristiche precise che differenziano un tipo dall'altro e che ne ostacolano la lettura da parte di dispositivi elettro-ottici. In figura si notano alcuni segni che distinguono un carattere dall'altro; se il confronto venisse fatto su basi più generiche, cercando di ampliare la gamma di caratteri riconoscibili, si rischierebbe di ottenere il risultato opposto ad esempio: le q e le g potrebbero essere facilmente confuse tra loro

te. Durante lo scorrimento del lettore, l'Omni-Reader considera un carattere soltanto quando esso si trova interamente sotto il captatore, ossia ben centrato e delimitato. Solo così è possibile una corretta scansione orizzontale e verticale dell'immagine ai fini del confronto e del riconoscimento.

Quando e se viene trovata una corrispondenza, l'Omni-Reader, tramite la porta seriale RS232, invia il codice ASCII equivalente al computer, che provvede a visualizzarlo sullo schermo. Naturalmente, la lettura avviene regolarmente soltanto se il testo è ben allineato con la finestrella di lettura. Ciò vale in particolare per le gambette di alcune lettere: se escono parzialmente dalla finestrella di lettura, il carattere può essere male interpretato (ad esempio , una p può diventare una o). Inoltre, se la riga di testo non è ben posizionata, può accadere che il lettore capti sezioni di caratteri appartenenti alla riga superiore o inferiore. I fabbricanti dell'Omni-Reader, del resto, non garantiscono la lettura di caratteri il cui "corpo" non sia compreso tra i 10 e i 12 punti tipografici (quattro/cinque caratteri per centimetro).

Seguendo gli accorgimenti fin qui esposti, l'Omni-Reader dimostra di essere un eccellente mezzo di acquisizione dati, che può far risparmiare notevoli quantità di tempo. La scansione di una riga di testo, su fogli nel formato A4, richiede in media tra 0,5 e 1,5 secondi. Non occorre certo lavorare con il cronometro in mano: i tempi indicati corrispondono all'incirca ad uno scorrevole movimento della mano sul foglio. Inizialmente si possono avere difficoltà nell'allineamento delle righe di testo, ma anche una persona poco esperta riesce ad imparare l'uso dell'Omni-Reader nel giro di mezz'ora.

Opzioni e comandi

Per agevolare la lettura di testi dai contorni non molto netti e precisi, sono previste alcune opzioni. La scelta di queste avviene "passando" il lettore su comandi scritti (quattro di essi si trovano sulla tavoletta, gli altri nel manuale). L'immissione dei comandi va preceduta dalla lettura di due speciali quadratini, affinché il dispositivo sia in grado di riconoscerli come tali e non come testo da inviare allo schermo.

Dei quattro comandi riprodotti sulla tavoletta, quello di uso più frequente è il TYPEFACE, che permette di cambiare il set dei caratteri da usare come campione. L'opzione NUMERIC rimuove dal set di caratteri campione tutte le lettere e i simboli, mantenendo soltanto i numeri. Questa opzione è utilissima, poiché spesso è proprio nella trascrizione manuale di cifre che si commettono più errori. Un'immissione diretta di questo genere evita, ad esempio, le così frequenti trasposizioni tra due cifre contigue.

Con l'opzione POORCOPY si può tentare di leggere un testo la cui qualità di stampa non sia ottimale. In questo caso la velocità di lettura risulta necessariamente più lenta del normale, per consentire al dispositivo un accurato esame di ciascun carattere del testo. Questa opzione è molto utile quando si tratta di leggere testi prodotti da una macchina da scrivere con nastro inchiostrato molto consunto.

Il software dell'Omni-Reader consiste di due programmi di controllo che risiedono nell'area di sistema della memoria. Ciò significa che, contemporaneamente ad essi, in memoria possono essere caricati programmi applicativi. Di consegenza, il testo letto dall'Omni-Reader può essere immesso direttamente in un file di un word-processor oppure di un programma "spreadsheet" (magari utilizzando l'opzione NUMERIC).

Il package ha un costo relativamente alto per applicazioni dilettantistiche, ma abbastanza ragionevole, se si considerano i vantaggi, per il settore commerciale dove la trascrizione di corrispondenza o perlomeno di materiale stampato è un'esigenza quotidiana.

Il FORTH

Un linguaggio nato, inizialmente, per il controllo di un radiotelescopio

Sviluppato verso la fine degli anni '60 dall'astronomo Charles Moore, il FORTH è stato progettato per controllare strumenti scientifici il cui posizionamento va effettuato con grande precisione. Sebbene gli scopi del FORTH non si limitino ad applicazioni di controllo, per comprenderne meglio i principi di funzionamento conviene proprio esaminare come viene impiegato per controllare i movimenti di un telescopio usando il terminale di un computer di un osservatorio.
Impartendo, ad esempio, il comando:

ELEVAZIONE 30 GRADI

il telescopio assume tale angolazione con rapidità e precisione. Inoltre, il sistema dovrebbe consentire di programmare e memorizzare sequenze di comandi simili al precedente, che possano essere richiamati ed eseguiti più volte. Se la posizione di "parcheggio" del telescopio ha un'elevazione di 90° e un azimuth di 0° (la nomenclatura è quella che indica l'angolazione e l'orientamento del telescopio), si deve poter definire un nuovo comando PARC del tipo:

AZIMUTH 0 GRADI ELEVAZIONE 90 GRADI

In sintesi, il sistema di controllo corrisponde a un completo linguaggio di programmazione del telescopio basato su:

- Comandi elementari di posizionamento.
- Funzioni per la creazione di comandi più complessi ("programmi" del telescopio), composti da comandi elementari.

Tuttavia il sistema deve possedere alcune particolari proprietà: deve poter accettare direttamente dalla tastiera ed eseguire nel modo più semplice possibile sia comandi "interni" come ELEVAZIONE sia comandi nuovi. Si supponga di risolvere il primo problema ricorrendo ad alcuni comandi interni del BASIC come PRINT e RUN; resta tuttavia quello dei nuovi comandi.
Immaginando di aver già definito AZIMUTH e ELEVAZIONE con due routine chiamate da istruzioni GOSUB alle linee 1000 e 1100, è possibile programmare il nuovo comando PARC nel modo seguente:

```
1200 REM PARC
1205 LET GRADI = 0
1210 GOSUB 1000
1215 LET GRADI = 90
1220 GOSUB 1100
1225 RETURN
```

oppure definire a priori una variabile PARC con l'istruzione di assegnazione:

```
LET PARC = 1200
```

e successivamente inserire il comando (valido in alcune versioni del BASIC):

```
GOSUB PARC
```

Tuttavia il miglior modo di procedere è probabilmente definire la routine GOSUB in forma di procedura (come avviene nel BASIC del BBC) introducendo l'istruzione:

```
PROCPARC
```

In questo modo, per chiamare le diverse routine di controllo del telescopio, è sufficiente digitare i comandi GOSUB o PROC, seguiti dal nome o dal numero di linea della routine desiderata.
Una tecnica alternativa richiede la scrittura di un programma "interprete", che prenda in carico i comandi del telescopio e li esegua. Ad esempio:

```
100 INPUT C$
150 IF C$ = "PARC" GOSUB 1200: GOTO 100
```

Purtroppo questo metodo non consente di introdurre direttamente i comandi BASIC (come PRINT).



IAN McKINNELL

Un computer... alternativo

Il "Jupiter Ace" rappresenta un coraggioso tentativo di offrire il FORTH come linguaggio standard al posto del BASIC. Sfortunatamente questo computer ha riscosso scarso successo per la feroce concorrenza dello Spectrum Sinclair, per la limitata popolarità del linguaggio FORTH e per le ridotte dimensioni della memoria a disposizione delle funzioni di colore. Tuttavia una nuova versione dell'apparecchio, corredata della documentazione originale (che era di ottima qualità), potrebbe ancora rappresentare un'eccellente proposta per lo studio del FORTH

Proprietà fondamentali del FORTH

Esaminiamo adesso le due proprietà principali del linguaggio di controllo del radiotelescopio:

- Il linguaggio deve essere "interattivo", deve cioè eseguire immediatamente i comandi inseriti da tastiera. Questa caratteristica, comune ai linguaggi BASIC, LOGO, APL e PROLOG e ai comandi dei sistemi operativi come il CP/M e Unix, è invece assente in PASCAL, ALGOL, FORTRAN e COBOL, essendo questi linguaggi "compilati".

- Il linguaggio deve essere "estendibile". Il formato delle chiamate per i nuovi comandi creati deve essere identico a quello per i comandi interni. Quindi le routine definite dall'utente devono essere considerate come parti integranti di un linguaggio esteso e più potente e non come nuove routine aggiunte al linguaggio originale. Questo secondo principio vale per il LOGO ed i sistemi operativi, ma non per gli altri linguaggi.

Il concetto di estendibilità è, in realtà, molto più complesso di quanto si potrebbe immaginare. È infatti possibile trasformare un comune "linguaggio di controllo per telescopio di tipo generico" in un "linguaggio di controllo esteso e personalizzato" dotandolo di routine di complessità crescente, appositamente progettate per controllare il proprio telescopio o altre apparecchiature collegabili a un computer, come robot, strumenti per sperimentazione scientifica, piccole linee di produzione e così via. Inoltre l'oggetto controllato dal linguaggio non deve necessariamente essere di tipo "fisico": va bene anche una qualsiasi applicazione "astratta" gestita da un programma, come una tartaruga in movimento su uno schermo grafico o i record di un sistema d'archiviazione.
Il FORTH è quindi lo strumento di programmazione che più si avvicina a un ideale "linguaggio del computer estendibile e universale".

DIALETTI FORTH

Le principali versioni del FORTH sono tre: il fig (FORTH Interest Group) FORTH, il FORTH-79 e il FORTH-83. A questi si conformano la maggior parte dei dialetti oggi in uso.
Il FORTH-79 e il FORTH-83 sono "versioni standard" successive e offrono una serie di comandi e di routine di base. Naturalmente i numerosi dialetti possiedono una dotazione di comandi più vasta, ma i programmi scritti in versione FORTH-83 standard possono essere eseguiti su qualsiasi dialetto FORTH-83, anche se questi non possiedono tutte le funzioni disponibili. Il FORTH-83 è l'ultima versione del linguaggio: presenta solo lievi incompatibilità con la precedente, mentre invece possiede una serie di funzioni interne supplementari. Molti dialetti del FORTH-79 e del FORTH-83 sono in realtà versioni del figFORTH adattate allo standard corrente

IL TELESCOPIO E LA TARTARUGA

I linguaggi FORTH e LOGO presentano notevoli somiglianze perché entrambi vennero originariamente progettati per controllare due apparecchiature, un telescopio e una tartaruga da pavimento rispettivamente. Questi linguaggi hanno in comune due moduli di funzionamento fondamentali: le routine vengono chiamate digitandone il nome e le nuove routine vengono definite usando quelle esistenti.
Tuttavia sono le stesse aspettative degli utenti che hanno contribuito a differenziare il FORTH dal LOGO: i programmatori FORTH puntano alla velocità d'esecuzione (quindi il linguaggio, oltre a essere interattivo e estendibile, offre anche un'elevata rapidità d'elaborazione) e sono disposti ad affrontre notevoli difficoltà pur di raggiungere il risultato desiderato. Viceversa il LOGO è destinato a semplici applicazioni di tipo didattico, che non richiedono un'alta velocità d'esecuzione.
Confrontando i due linguaggi, si nota che il FORTH è privo della grafica per la tartaruga e possiede funzioni di basso livello per la gestione dello schermo. Queste possono essere adoperate per definire "parole" (tipiche della tartaruga), quali FORWARD, RIGHT ecc., ossia per creare una versione estesa del FORTH che comprenda anche quel tipo di grafica. Ad esempio, confrontando le istruzioni usate dai due linguaggi per controllare la tartaruga:

LOGO	FORTH
FORWARD 100	100 FORWARD

si vede che il FORTH sembra procedere in senso inverso. Ciò accade perché i parametri forniti a una routine devono essere calcolati e inseriti nel comando prima del nome della routine.
Inoltre i due linguaggi possiedono differenti comandi per la definizione delle nuove routine:

TO... END	: ... ;

e delle strutture iterative:

REPEAT 4[...]	40 DO ... LOOP

Come esempio, usiamo i precedenti comandi per definire la procedura QUADRATO:

```
TO QUADRATO LATO          :QUADRATO
REPEAT 4[                   40 DO
  FORWARD LATO                DUP FORWARD
  RIGHT 90                    90 RIGHT
  ]                         LOOP
                            DROP
END                       ;

QUADRATO 100              100 QUADRATO
```

I comandi DUP e DROP "manipolano" il contenuto dello stack consentendo alle routine di ricevere il parametro di QUADRATO (DUP crea una copia del parametro per FORWARD mentre DROP la elimina dallo stack per evitare che venga conteggiata nel calcolo successivo)



MARCUS WILSON-SMITH





Pacchetti FORTH

Il FORTH ha suscitato un notevole interesse tra gli appassionati di home computer. Per questo motivo, molte software house hanno prodotto pacchetti FORTH per i più popolari home computer. Tra questi: il FORTH della Acornsoft (disponibile anche in ROM) e lo SkyWays Multi-FORTH per il Micro BBC; il FORTH Abersoft e il FORTH Artic per lo Spectrum Sinclair

Ultime routine ROM

Nuovi comandi BASIC e alcuni trucchi per una migliore gestione del video dello Spectrum

Intercettatore d'interfaccia

Quando l'Interface 1 è collegata, lo Spectrum reagisce ad un errore eseguendo una chiamata RST #08, che provoca l'impaginamento della ROM "ombra" (contenuta, appunto, nell'Interface 1) e l'esecuzione della routine il cui indirizzo è depositato nella variabile di sistema VECTOR. Questo meccanismo consente di variare l'indirizzo in VECTOR, facendo così eseguire una routine scritta dall'utente ogni volta che viene riscontrato un errore



Qualsiasi nuovo comando si desideri aggiungere al BASIC dello Spectrum, esso viene considerato un errore di sintassi sia dalle routine di verifica normali, nelle ROM del computer, sia da quelle situate nella ROM dell'Interface 1. Solo in tal caso, infatti, il controllo passa alla routine il cui indirizzo si trova nella variabile di sistema VECTOR (locazioni 23735 e 23736). Per far ciò si possono seguire due strade:

1. Modificare una parola chiave già esistente, in modo che essa non venga riconosciuta valida. L'inclusione di un numero errato di parametri oppure l'aggiunta di un carattere estraneo (chiamato "breaker") provoca l'innesco di un errore di sintassi.
   Ad esempio:

   LINE 10, 10, 1, 19 (Troppi parametri)
   BORDER * (L'asterisco è un codice estraneo)

2. Aggiungere un comando completamente nuovo usando, ad esempio, un codice estraneo come prefisso a un comando preesistente, come in: *BEEP oppure *PRINT.

L'accesso di questi nuovi comandi alle variabili del BASIC non differisce da quello normale. Qui vengono creati nuovi comandi utilizzando il secondo metodo citato.
Vediamo come aggiungere il comando *B, da usare per l'emissione di una nota per la durata di un secondo. Per prima cosa occorre decidere dove, in memoria, collocare la routine relativa al nuovo comando. Ricorrere al ben noto stratagemma di sfruttare lo spazio accantonato con frasi REM non conviene, poiché l'indirizzo dei dati contenuti in un programma BASIC varia, se al sistema viene collegata l'Interface 1. È preferibile invece impiegare il comando CLEAR nn, col quale si può riservare tutto lo spazio che si desidera. Qualunque sia il comando da aggiungere, occorre includere nella struttura del programma i passi descritti nel diagramma a fianco. Quando l'interprete BASIC arriva a chiamare la routine il cui indirizzo si trova in VECTOR, è necessario estrarre il carattere del programma BASIC che ha provocato l'errore, per vedere se si tratta effettivamente del nuovo comando aggiunto. Il puntatore al carattere in questione si trova nella variabile di sistema CHADD (locazioni 23755 e 23756).
Per estrarre il carattere puntato da CHADD si possono adoperare due routine della ROM principale: GETCHAR e NEXTCHAR. Di queste routine basta sapere che entrambe possono essere richiamate quando è abilitata la ROM ombra usando l'istruzione RST #10, seguita dal codice appropriato (rispettivamente #0018 e #0020).
Entrando nella routine puntata da VECTOR, CHADD contiene l'indirizzo del primo carattere che ha provocato l'errore, quindi l'esecuzione del nuovo comando *B implica l'uso di una routine di questo tipo:

```
D7          vector:  rst   #10
1800                 defw  #0018        ;codice per GETCHAR
FE2A                 cp    "*"          ;è un "*"?
C2F001               jp    nz,#01F0     ;no -> errore
D7                   rst   #10
2000                 defw  #0020        ;codice di NEXTCHAR
FE42                 cp    "B"          ;è una "B"?
C2F001               jp    nz,#01F0     ;no -> errore
D7                   rst   #10          ;Il carattere successivo deve
2000                 defw  #0020        ;confermare la fine del comando
CDB705               call  #05B7        ;verifica la fine del comando
            esecuz:
```

La routine all'indirizzo #05B7 della ROM ombra va chiamata previo caricamento nel registro A del carattere interessato, puntato da CHADD. Si potrebbe anche adoperare una nuova chiamata a NEXTCHAR. Tuttavia, la routine #05B7 ha un comportamento diverso: in sede di verifica della sintassi il ritorno di tale routine è alla ROM principale (per proseguire con l'esame del successivo comando), mentre in sede d'esecuzione il controllo viene restituito al codice chiamante, pertanto viene eseguito tutto ciò che segue l'etichetta esecuz.
Si noti che il meccanismo di verifica può essere ampliato, per consentire la creazione di nuovi comandi composti anche da lettere minuscole o da combinazioni di maiuscole e di minuscole. Se un carattere non corrisponde alla sequenza del comando, l'uscita dalla routine deve avvenire tramite un salto alla routine #01F0 della ROM ombra (versione 1), per segnalare l'errore. Si noti anche che, all'ingresso nella routine puntata da VECTOR, va estratto il carattere corrente del comando in esame, tramite GETCHAR (indirizzo #0018), anziché NEXTCHAR (indirizzo #0020).
La sezione esecutiva della routine per il nuovo comando è molto semplice e ricorre alla normale routine di "beep" della ROM principale:

```
110501    esecuz:  ld    de,#0105    ;durata = 1 secondo
21007              ld    hl,#0700    ;altezza della nota
D7                 rst   #10
B503               defw  #03B5       ;routine BEEP
C3C105             jp    #05C1       ;uscita
```

Il salto all'indirizzo #05C1 serve per una corretta conclusione della routine, restituendo il controllo alla ROM principale.
Ovviamente, un nuovo comando può sfruttare una qualsiasi delle routine fin qui esaminate, esistenti nella ROM principale e, inoltre, anche quelle relative alla grafica (descritte più avanti).
Il metodo di accesso è sempre il solito: l'istruzione RST #10 seguita dall'opportuno codice.
Per fornire un esempio di ciò, ecco una routine che effettua la valutazione di un parametro specificato dall'utente, usando le due routine di sistema #1C82 e #1E94. Per semplicità il nome del comando è sempre ∗B, ma il suo formato è adesso ∗B n, dove n è un qualsiasi valore compreso tra 0 e 255. Tale parametro indica la durata del "beep" in secondi, cosicché ∗B 255 genera una nota per 255 secondi.

```
                org   60000
        VECOTR  equ   #5CB7          ;indirizzo di VECTOR
CF      iniz:   rst   #8
31              defb  49             ;preparazione variabili IF1
2169EA          ld    hl,newcom      ;indirizzo nuovo comando
22B75C          ld    (VECTOR),hl    ;depositato in VECTOR
C9              ret                  ;fine inizializzazione
        :
D7      nuovo   :rst  #10            ;VECTOR punta qui
1800            defw  #0018          ;chiamata a GETCHAR
FE2A            cp    "*"
C2F001          jp    nz,#01F0
D7              rst   #10
2000            defw  #0020          ;chiamata a NEXTCHAR
FE42            cp    "B"
C2F001          jp    nz,#01F0
D7              rst   #10
2000            defw  #0020
D7              rst   #10
821C            defw  #1C82          ;valutazione del parametro
CDB705          call  #05B7          ;controllo di fine comando
D7      esecuz: rst   #10
941E            defw  #1E94          ;val. parametro in A
47              ld    b,a
C5      ciclo:  push  bc             ;salva contatore
110501          ld    de,#0105       ;durata
210007          ld    hl,#0700       ;altezza
D7              rst   #10
B503            defw  #03B5          ;chiamata a BEEP
C1              pop   bc             ;recupera il contatore
10F3            djnz  loop
C3C105          jp    #05C1          ;tutto fatto
```

Si noti che, siccome la routine #1C82 accetta soltanto valori a otto bit (passati nel registro A), numeri esterni all'intervallo 0-255 generano un errore. Un'altra cosa da notare è che, durante l'emissione della nota, la variabile FRAMES non viene incrementata e non viene eseguita la scansione della tastiera: specificare il valore 255 blocca l'uso del computer per oltre quattro minuti!

Ulteriori parametri

Come accennato in precedenza, si possono aggiungere nuovi comandi specificando semplicemente un numero di parametri diverso da quello previsto da normali comandi BASIC, mantenendone il nome originale. Tutti i seguenti esempi provocano un salto all'indirizzo puntato da VECTOR:

```
CIRCLE x,y,z,n
LINE x
BORDER x,y,z
```

IDENTIKIT DELL'INTERFACE 1

Per adoperare proficuamente le routine della ROM ombra contenuta nell'Interface 1, è necessario per prima cosa verificare di quale versione ROM si tratta? Come regola approssimativa, tutte le Interface 1 con numero di serie superiore a 87315 possiedono la versione 2 della ROM, ma una conferma si ha usando:

```
CLOSE #0:PRINT PEEK 23729
```

Il valore visualizzato è 0 per la versione 1 e 80 per la versione 2. In ogni caso, le routine descritte nelle varie parti dedicate al SO dello Spectrum hanno indirizzi identici nelle due versioni. Volendo adoperare anche altre routine, il modo migliore di procedere è quello di usare un "disassemblatore" in combinazione con il programma per l'impaginamento della ROM ombra, presentato a pag. 1900.
Nella versione 2, oltre ad altre modifiche, sono stati aggiunti due nuovi codici di aggancio: #33 che estrae un descrittore di record per file del Microdrive, e #34 che apre un canale "b" sulla porta seriale RS232 (la routine abbinata al codice #34 deposita l'indirizzo iniziale del canale nel registro doppio DE)

INDIRIZZI UTILI

Nella ROM ombra:

#01F0	Uscita dalla ROM ombra tramite la routine d'errore della ROM principale
#05B7	Ritorna normalmente, durante l'esecuzione di programmi, se il carattere nel registro A è #0D (ENTER) oppure un "due punti", altrimenti genera un errore
#05C1	Ritorna all'interprete principale quando è stato accettato un nuovo comando

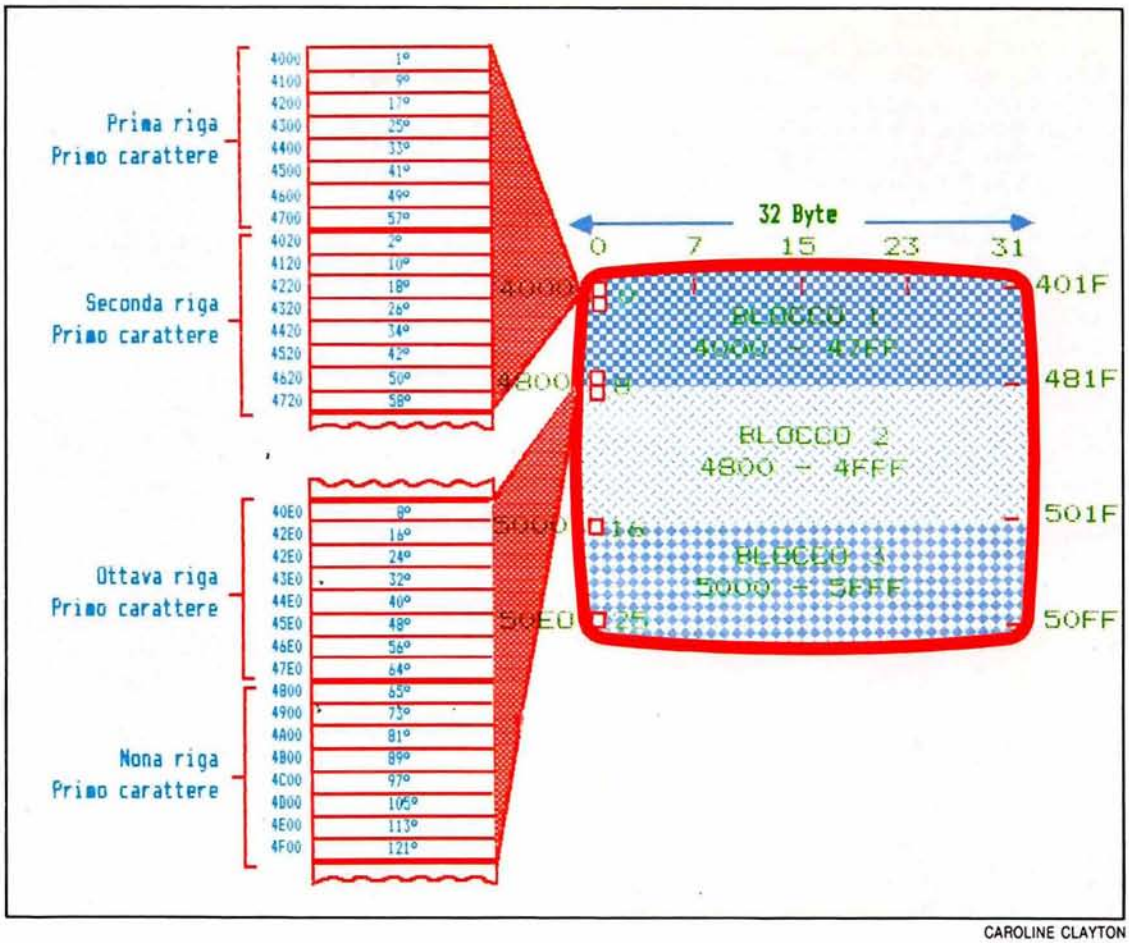
Nella ROM principale:

#0018	Routine GETCHAR. Deposita in HL il valore di CHADD (vedere sotto) e copia nell'accumulatore A il carattere puntato da CHADD. Inoltre verifica se il carattere è visualizzabile. In tal caso ritorna normalmente, altrimenti passa a NEXTCHAR
#0020	Routine NEXTCHAR. Incrementa il valore di CHADD e lo deposita in HL. Se il carattere puntato da CHADD è visualizzabile esso viene copiato nell'accumulatore A e la routine ritorna. In caso contrario il processo è ripetuto da capo
#1C82	Valuta o verifica un'espressione numerica, il cui primo elemento deve essere puntato da CHADD. In uscita, CHADD punta al primo carattere che segue l'espressione, mentre il risultato è depositato sulla sommità dello stack del calcolatore ROM
#1E94	Estrae un valore dalla sommità dello stack del calcolatore e ritorna depositandolo nel registro A, purché questo sia compreso tra 0 e 255. In caso contrario viene generato un errore

La variabile di sistema CHADD punta al corrente carattere in esame ed è situata in 23755 e 23756

Memoria video

Questo diagramma mostra la struttura globale del file del display per lo Spectrum. Lo schermo è suddiviso in tre blocchi principali, ognuno dei quali contiene 256 posizioni di carattere (otto righe di 32 caratteri). All'interno di ciascun blocco le righe di pixel sono disposte nell'ordine qui riportato: il primo blocco di 256 byte contiene le informazioni per la prima riga di pixel dei primi 256 caratteri, il secondo blocco le informazioni per la seconda riga e via dicendo. I dettagli di questa struttura si possono meglio apprezzare nella parte sinistra della figura. Per ogni riga di pixel in un carattere è indicato (1) il relativo indirizzo nel file del display e (2) l'ordine col quale vengono riempite le righe di pixel quando si eseguono POKE consecutive nel file del display



CAROLINE CLAYTON

Entrando nella rotuine puntata da VECTOR, CHADD contiene l'indirizzo del codice (o "token") corrispondente al comando in cui è stato riscontrato un errore. Tale codice, ovviamente, va confrontato con quello interessato, desumibile dal manuale dello Spectrum. L'aggiunta di un nuovo comando al BASIC dello Spectrum è abbastanza semplice, nonostante l'apparente complessità delle operazioni. L'importante è ricordarsi quale ROM è abilitata durante le varie fasi. Se, per qualsiasi motivo, si esce dalla ROM ombra, basta usare il codice di aggancio 50 (e relativo indirizzo) per chiamare routine della ROM ombra.

Le routine ROM per la grafica

La gestione dello schermo, sullo Spectrum, non presenta difficoltà di rilievo. Per primo viene preso in esame il metodo che ricorre alle routine ROM usate dal sistema operativo del computer.

Per accedere allo schermo, occorre innanzi tutto aprire l'apposito canale S, selezionando il flusso 2. Per la normale visualizzazione di caratteri, a questo punto, la via più pratica è quella di usare RST #10. Tuttavia, volendo visualizzare un'intera stringa di caratteri alla volta, è necessaria una breve routine che estragga uno ad uno i caratteri e li visualizzi tramite RST #10.

Tale routine (chiamata Print—String) esiste già nella ROM del computer, all'indirizzo #203C. Per usarla basta caricare nella coppia di registri DE l'indirizzo iniziale della stringa e in BC la sua lunghezza.

Ecco un listato che ne dimostra l'uso:

```
                ORG  60000
1172EA          LD   DE,MSG
3E02            LD   A,2              ;seleziona il flusso 2
CD0116          CALL #1601            ;apre il canale S
1172EA          LD   DE,MSG           ;DE punta alla stringa
012600          LD   BC,38            ;BC contiene la lunghezza
CD3C20          CALL #203C            ;chiamata a PRINT-STRING
C9              RET
160A06  MSG:    DEFB 22,10,6          ;codici per AT 10,5
494C204D        DEFM "IL MIO COMPUTER"
160C07          DEFB 22,12,7
434F5253        DEFM "CORSO PRATICO"
```

Esiste anche un'altra utile routine ROM, per la visualizzazione di numeri a virgola fluttuante (chiamata PRINT—FP), il cui indirizzo è #2DE3. L'uso dello stack e della routine a #1C82, durante i calcoli numerici, è già stato descritto (vedere a pag. 1910). La routine PRINT—FP rende certamente le cose più semplici, perché doversi occupare in dettaglio della visualizzazione di valori numerici è sempre un compito noioso. Innanzi tutto, per visualizzare un numero, è necessario depositarlo sullo stack del calcolatore, usando il doppio registro BC oppure il registro A. Le rispettive routine STACK—BC e STACK—A si trovano agli indirizzi #2D2B e #2D28. A questo punto basta chiamare la routine PRINT—FP:

```
                ORG  60000
013930          LD   BC,12345         ;carica il numero in BC
```

```
CD2B2D      CALL  #2D2B       ;e lo deposita sullo stack
3E02        LD    A,2
CD0116      CALL  #1601       ;apre il canale S
CDE32D      CALL  #2DE3       ;visualizza il numero
C9          RET
```

La potenza grafica dello Spectrum, tuttavia, risulta più evidente adoperando i comandi ad alta risoluzione PLOT e DRAW, il primo dei quali è già stato esaminato in precedenza (vedere a pag. 1797). L'uso di DRAW è altrettanto facile: basta specificare qualche parametro in più. DRAW tenta di tracciare la linea più diritta tra il punto selezionato dall'ultimo PLOT e quello che viene definito nel comando in base all'origine. Se, per esempio, la linea deve congiungersi con un punto situato 20 pixel verso l'alto e 40 pixel a sinistra, il comando da usare è DRAW −40,20.

Per spiegare l'impiego di DRAW in codice macchina, viene adottata la sintassi DRAW x,y, dove x e y rappresentano due parametri da considerare separatamente. Per semplificare le cose, vengono per il momento ignorati eventuali fattori di curvatura. Un necessario accorgimento, nell'uso di DRAW in codice macchina, è quello di preservare il contenuto della coppia di registri H'L', ripristinandolo subito prima di tornare al BASIC.

Il valore assoluto di x (ABS x) è passato nel registro C, ABS y nel registro B mentre i segni di x e y rispettivamente nei registri D ed E. Questo metodo consente il passaggio anche di valori per il segno diversi da −1, 0 e 1, che vengono sommati alla posizione dell'ultimo pixel "plottato", per ottenere quella del pixel successivo.

Pertanto, usando ad esempio un valore di segno pari a 2 o 3, la linea risulta due o tre volte più lunga del normale, con un effetto punteggiato simile a quello che si ottiene sul QL Sinclair.

Si noti che il valore del segno va sempre espresso in "complemento a due". Esempi di entrambi i criteri di tracciamento si possono vedere nel seguente listato:

```
            ORG  60000
D9          EXX
E5          PUSH HL           ;salva H'L'
D9          EXX
0658        LD   B,88         ;coordinata Y
0E64        LD   C,100        ;coordinata X
CDE522      CALL #22E5        ;PLOT 100,88
0E28        LD   C,40         ;ABS x=40
0614        LD   B,20         ;ABS y=20
1EFF        LD   E,255        ;SGN y=−1
1601        LD   D,1          ;SGN x=1
CDBA24      CALL #24BA        ;DRAW −40,20
3E58        LD   A,88
327D5C      LD   (23677),A    ;X org=88
3E32        LD   A,50
327E5C      LD   (23678),A    ;Y org=50
0E28        LD   C,40
0614        LD   B,20
1602        LD   D,2          ;SGN x=2 (linea punteggiata)
1E02        LD   E,2          ;SGN y=2 (linea punteggiata)
CDBA24      CALL #24BA
D9          EXX
E1          POP  HL           ;ripristina H'L'
D9          EXX
C9          RET
```

Le informazioni nelle due tabelle presentate più avanti riassumono tutto quanto concerne l'uso della grafica in codice macchina fin qui esaminato. La prima tavola contiene un elenco delle routine più utili, mentre la seconda riporta un elenco delle variabili di sistema interessate.

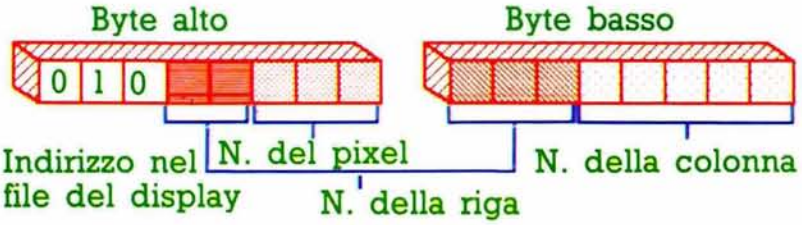
Il file del display

La memoria video dello Spectrum è suddivisa in due sezioni. La prima, chiamata "file del display", contiene le informazioni sui singoli pixel. La seconda, detta "file degli attributi", contiene invece le informazioni di colore per lo schermo. Il file del display, 6144 byte in tutto, è compreso tra gli indirizzi 16384 e 22527. Si trascriva e si esegua il seguente programma:

```
10 FOR F=16384 TO 22527
20 POKE F,255
30 NEXT F
```

Gradualmente, durante l'esecuzione, lo schermo si riempie, ma da un attento esame si può notare che tale riempimento avviene in modo non sequenziale. In altre parole, dopo aver tracciato la prima riga di pixel, la successiva non è subito sotto ma dista otto pixel verso il basso. Raggiunta la sessantaquattresima riga, il processo riparte dalla seconda riga in alto sullo schermo, seguita dalla nona e così via. Tutto ciò prosegue finché non si riempie il terzo superiore dello schermo, quando il processo riparte per il terzo centrale dello schermo e, infine, per quello inferiore (vedere figura in basso).

Per adoperare il file del display (abbreviato in Dfile) e calcolare gli indirizzi da usare per accedere ai pixel, si consideri il seguente diagramma:



Indirizzi nel Dfile
Questo diagramma illustra il significato dei bit nelle locazioni all'interno del file del display. Sebbene apparentemente complessa, la struttura di questa parte della memoria video consente una rapida elaborazione dei dati da visualizzare, ricorrendo ad operazioni su singolo registro in codice macchina

Questo mostra come sono disposti i bit negli indirizzi del Dfile, con ciascun numero di pixel che si riferisce alla riga di pixel all'interno di ogni carattere. Il pixel più in alto a sinistra di un carattere è il pixel 0.

Questo diagramma mostra anche che volendo spostarsi di un pixel verso il basso, sempre all'interno di un carattere, basta incrementare il registro H con INC H. Per muoversi invece di un carattere verso destra, si incrementa la coppia HL con INC HL.

Per calcolare una posizione da usare come coordinata per una PRINT AT, si ricorre al programma listato qui sotto. Nel registro E va depositato il numero della colonna e nel registro D il numero della riga. Dopo una chiamata a questa routine, il doppio registro HL contiene l'indirizzo corretto.

```
            ORG  60000
3EAF        LD   A,175
92          SUB  D
```

```
57      LD   D,A        ;D contiene 175 (coord. y)
A7      AND  A
1F      RRA
37      SCF
1F      RRA
A7      AND  A
1F      RRA             ;scorrimento dei bit
AA      XOR  D
E6F8    AND  248
AA      XOR  D
67      LD   H,A        ;H diventa...
7B      LD   A,E        ;64 + 8*INT(B/64) + B(MOD 8)
CB07    RLC  A
CB07    RLC  A
CB07    RLC  A
AA      XOR  D
CB07    RLC  A
CB07    RLC  A          ;scorrimento dei bit
6F      LD   L,A        ;L vale 32*INT(...
7B      LD   A,E        ;...B(MOD64)/8) + INT(x/8)
E607    AND  7          ;il byte basso è x(MOD 8)
C9      RET
```

Comunque, volendo calcolare l'effettivo indirizzo di un pixel occorre un programma diverso (anche se simile a questo per quanto concerne il passaggio dei parametri):

```
        ORG  60000
7A      LD   A,D
E6F8    AND  248        ;"maschera" i bit 0-2
CBF7    SET  6,A
67      LD   H,A
7A      LD   A,D
E607    AND  7          ;"maschera" i bit 3-7
CB0F    RRC  A
CB0F    RRC  A
CB0F    RRC  A          ;scorrimento dei bit
83      ADD  A,E        ;somma i bit della colonna
6F      LD   L,A
C9      RET
```

Anche in questo caso D ed E contengono rispettivamente le coordinate y e x, mentre in uscita HL contiene l'indirizzo del pixel (con valore compreso tra 0 e 7).
Un ultimo fatto da ricrdare è che eseguire la POKE di un byte nel Dfile significa infuenzare 8 pixel alla volta.
Questi pixel sono sempre disposti come quelli che si riferiscono alla grafica definita da utente.

Il file degli attributi

La seconda sezione della memoria video, ossia il file degli attributi, inizia da 22528 ed ha una lunghezza di 768 byte. La struttura di questo file è molto semplice. Si esegua il seguente programma:

```
10 FOR F=22528 TO 22528+767
20 POKE F,0
30 NEXT F
```

Si noti che lo schermo viene gradualmente riempito in modo normale: una riga dopo l'altra e dall'alto verso il basso.
Ogni byte contiene gli attributi di INK, PAPER, BRIGHT e FLASH relativi a ciascun carattere dello schermo. Per la precisione, i primi tre bit si riferiscono al colore di INK, i bit 3, 4 e 5 al colore di PAPER. Il bit 6 contiene l'appropriato valore di BRIGHT e il bit 7 quello di FLASH.
In ultimo, ecco come cambiare il colore di BORDER: si carichi il numero del colore nel registro A e poi si chiami la routine all'indirizzo esadecimale #229B.

VARIABILI DELLO SPECTRUM

Nome	Locazione	Descrizione
COORDS	23677	Contiene la coordinata x dell'ultimo punto di plot
COORDS	23678	Contiene la coordinata y dell'ultimo punto di plot
DF CC	23684	Questa è una variabile a due byte che contiene l'indirizzo nella memoria del file del display per il prossimo carattere da visualizzare
S POSN	23688	Numero di colonna della corrente posizione di visualizzazione
S POSN	23689	Numero di riga della corrente posizione di visualizzazione
ATTR P	23693	Usata per conservare i colori permanenti stabiliti da INK, PAPER, FLASH e BRIGHT. I dati sono nel formato ATTR
MASK P	23694	Usata per "mascherare" i valori di ATTR P, quando sono usati colori trasparenti (da INK 9 in poi). Ogni bit selezionato in modo da corrispondere ad ATTR P viene ignorato
ATTR T	23695	Come ATTR P, ma usato soltanto per i colori temporanei (nel caso di cambi di colore all'interno di una PRINT, ad esempio)
MASK T	23696	Come MASK P, ma per i colori temporanei
P FLAG	23697	Contiene vari "flag" che indicano lo stato dei diversi colori temporanei e permanenti. Ciascun bit indica se i relativi colori di ink o paper sono trasparenti oppure no

N.B. Per ulteriori descrizioni di ATTR P, MASK P e P FLAG, vedere a pag. 1797

ROUTINE ROM DELLO SPECTRUM

Nome	Locazione	Dati in ingresso	Risultati
RST #10	#0010	Il registro A deve contenere il codice ASCII del carattere	Visualizzazione del carattere
PR STRING	#203C	BC deve contenere la lunghezza della stringa e DE il suo indirizzo iniziale	La stringa viene visualizzata nella prima locazione di schermo libera
PRINTED FP	#2DE3	Il numero da visualizzare deve trovarsi sullo stack del calcolatore	Visualizzazione del numero situato in cima allo stack
STK-A	#2D28	Il registro A deve contenere il valore da appoggiare sullo stack	Il valore in A viene depositato sullo stack del calcolatore
STK-BC	#2D2B	Come STK-A ma per la coppia di registri BC	Il valore in BC viene depositato sullo stack del calcolatore
PLOT	#22E5	B e C devono contenere, rispettivamente, le coordinate y e x	Viene tracciato il punto (x,y) e aggiornata la variabile SV COOR
DRAW	#24BA	B deve contenere ABS(y) C deve contenere ABS(x) D deve contenere SGN(y) E deve contenere SGN(x)	Se i valori del segno sono superiori a 1 la linea è punteggiata
CHAN-OPEN	#1601	A deve contenere il numero di flusso da usare	Viene aperto il canale richiesto

INDICE GENERALE PER SETTORE

Primi appunti

VOLUME PRIMO

Lo sapevate che...? 13
Carta d'identità 16
Lo sapevate che...? 48
Lo sapevate che...? 64

Hardware

VOLUME PRIMO

Cos'è un computer 1
I più diffusi in Europa 6
Installare il micro 8
Lo Spectrum 17
Il Commodore 64 33
Dietro le... barre 49
L'Oric-1 50
I micro in movimento 65
Il Lynx 70
Il BBC modello B 89
Scienza domestica 106
Atari 400 e Atari 800 109
Il Dragon 32 130
Il Commodore Vic-20 150
L'era dei portatili 166
L'Epson HX-20 169
Il TI99/4A 189
Lo ZX81 210
Le qualità che contano 226
L'Aquarius 230
I computer analogici 238

VOLUME SECONDO

Il Sord M5 250
Il Tandy Color Computer 270
Il Memotech MTX 512 290
Memorie di oggi, memorie di domani 304
Lo Sharp MZ-711 309
Oltre i limiti 326
Il Tandy MC-10 330
L'Apple IIe 349
L'Electron della Acorn 370
Un arcobaleno di accessori 386
L'Olivetti M-10 390
L'Osborne-1 410
Il Commodore PET 4032 430
Il QL Sinclair 450
Il 380Z della Research Machines 470

VOLUME TERZO

La scelta migliore 484
Novità Commodore 490
La scelta di un buon monitor 509
I Microdrive Sinclair 514
Il chip Motorola 68000 523
Spectrum terza versione 530
Unità a dischi Comodore 532
Unità a dischi Atari 543
Metodi di stampa 550
Unità a dischi BBC 564
Il Personal dell'IBM 569
Unità a dischi della Dragon 584
L'AIM 65 della Rockwell 589
Incompatibilità dei dischi 604
Il Dragon 64 610
Lo Spectravideo 318 629
La Grafpad 649
Novità Atari 669
Lo Sharp PC-5000 690
Poligono di tiro 710

VOLUME QUARTO

Una gustosa albicocca 729
L'Atmos della Oric 749
Le stampanti/plotter 769
La grafica su stampante 784
Il Colour Genie 789
Varietà di stampa 804
TV contro monitor 809
Prima impressione 824
L'Advance 86a 829
Margherite per stampare 844
L'unità Torch per il BBC 849
L'Adam 869
La Brother EP-44 886
Dalla carta allo schermo 889
Il Prism VTX5000 906
L'Amstrad CPC 464 909
Il compressore di dati 921
L'interfaccia Acorn Plus 1 929
Il Macintosh 949

VOLUME QUINTO

L'Einstein 969
Il QL della Sinclair 981
Il Sega SC3000H 1009
Le tascabili della Casio 1021
L'Apple IIc 1048
Joypad senza fili 1070
Il PX-8 1089
Il Koala-pad 1109
Un trio da viaggio 1130
Il Toshiba HX-10 1149
Un giro tra le vetrine 1169
Il Plus/4 1189

VOLUME SESTO

Periferiche a confronto 1209
La Snap 1229
Il Beasty 1250
Il Commodore 16 1269
Lo Spectrum + 1286
La Touchmaster 1310
L'Osborne Encore 1329
Il Compaq Plus 1349
Il Link 480Z 1369
Foto computerizzate 1389
Due Sharp da taschino 1409
Memotech RS 128 1429

VOLUME SETTIMO

Il Music Maker 1441
Un computer da bambini 1450
Enterprise Sixty Four 1469
L'Apricot Portable 1489
Il CX5M della Yamaha 1509
Un programmatore di EPROM 1524
Il Mouse della AMX 1530
L'Electron Plus 3 1549
Piccoli robot 1564
Hybrid Music 500 1570
Il Sanyo MBC-550 1589
Il Rotronics Wafadrive 1609
Il Discovery 1 1534
Il Penman Plotter 1649
Il kit Fischertechnik 1669

VOLUME OTTAVO

Il DDI-1 della Amstrad 1689
Il Wren Executive 1709
Il Tandy 1000 1729
Stampanti per Spectrum 1741
La Banana Interface 1770
L'Atari 130XE 1789
Il Ninbus RM 1808
Il sintetizzatore "Echo" 1821
Il Quik Data Drive 1852

Indice generale per settore

L'Apricot F1e 1869
I supercomputer 1874
Il Magic Mouse 1884
Ottica e computer 1889
L'Omni-Read 1905

Software

VOLUME PRIMO

Computer e software 9
Il mondo dei giochi 14
L'artista elettronico 26
Dipingere coi numeri 44
Quando 1 + 1 = 10 54
La parola giusta 61
Appuntamento col chip 72
Quando meno vale più 79
Obiettivo Scuola 81
I diagrammi di flusso 104
Il "complemento a due" 119
Database: chi cerca... trova 124
La grafica "a sprite" 152
Gli "spreadsheet" 158
I giochi d'avventura 161
Interpreti e compilatori 184
I pirati del software 192
Organizzazione dei dati 204
Avversario: il computer 221

VOLUME SECONDO

Le liste concatenate 244
Il Lisa e le "icone" 261
Il comportamento dei modelli 267
I vari tipi di "sort" 286
Quattro bit... per un labirinto! 288
I "Bulletin Board" 306
Una marcia in più 328
I linguaggi del computer 344
Un po' d'immaginazione 366
Tre giochi di simulazione 389
Gli "spelling checker" 404
Programmi generatori specializzati 406
Elementi... sovversivi 432
Gli attrezzi del mestiere 444
Dpef Dsbdljoh 454
Rischio calcolato 461

VOLUME TERZO

Ant Attack: impariamo da un gioco 486
Il computer per insegnare 501
Software per divertimento 521
Esami al computer 541
L'ufficio computerizzato 592
Contabilità e computer 601
Tiriamo le somme 632
Ordini e magazzini 652
Memoria sotto controllo 664
Controllo sulle scorte 672
I file sequenziali 684
Resoconto finale 692
L'accesso ai file 706

VOLUME QUARTO

I file ad accesso casuale 724
Posizioni... chiave 752
Un gioco subacqueo 755
I file su cassette 774
Il Boogaboo 776
Gli Spreadsheet sono utili? 786
Il Manic Miner 793
Corrispondenza automatizzata 806
Twin Kingodom Valley 816
Apocalypse 836
Atic Atac 856
Programmare per vendere 861
Jet Pac 875
Snooker 895
Il Valhalla 936
Un tocco di... tastiera 943
Il River Rescue 953

VOLUME QUINTO

Il "Lords of Midnight" 975
Un po' di magia nera 997
Da un'idea al successo 1030
Classic racing 1080
Space Invaders 1095
La somma delle parti 1106
Pacman 1120
Una sinfonia corale 1124
Missile Command 1140
Un maggior controllo 1152
Star Raiders 1160
Una macchina da calcolo 1172
Mutui e prestiti 1192
Psytron 1200

VOLUME SESTO

Creare i modelli 1204
"Deus ex machina" 1220
Un piano d'azione 1232
Flyerfox 1240
Griglia d'acciaio 1244
Mugsy 1260
I Macro-comandi del Lotus 1264
Il TK!Solver 1284
Starfinder 1300
Indovinare ragionevolmente 1304
Il software verticale 1324
Summer games 1340
BrainStorm 1344
Potenza di calcolo 1372
Software e didattica 1386
Didattica d'oltremanica 1406
Il package "Human Edge" 1426

VOLUME SETTIMO

Il percorso critico 1464
Ghostbusters 1480
Il Project della Microsoft 1486
Leggere la Bibbia 1512
Elite 1520
Tartarughe... scolastiche 1534
Il computer... agricolo 1544
Zaxxon 1560
Organizzazione dei dati 1577
Usare un database 1584
Una "chiave" da cercare 1604
Massima immissione 1632
Ottime relazioni 1644
Database a doppio ingresso 1666
"Knight Lore" 1680

VOLUME OTTAVO

Il MicroPen 1687
Impossible Mission 1700
Scegliere il giusto DBM 1704
Archon 1740
Controllo dinamico 1744
I comandi "transienti" 1764
"Minder" 1780
I file nel CP/M 1784
"Shadowfire" 1800
I dischetti del CP/M 1804
Un futuro promettente 1826
Football Manager 1840
Codice che genera codice 1854
Ancora generatori di programmi 1872
Il DrLOGO 1895

Panorami

VOLUME PRIMO

Guida all'acquisto di un microcomputer 5
La nuova rivoluzione 24
L'insegnante elettronico 30
Il tasto giusto 36
La controrivoluzione 41
Lo scatto che conta 46
Velocità di risposta 56
Pronta cassa 60
Messaggio ricevuto! 66
Via con la stampa 74
Codice segreto 84
Dal pallottoliere al computer 86
La registrazione dei programmi 94
Le carriere nell'informatica 101
L'anello di congiunzione 108
Le unità a dischi 114
La fabbricazione dei chip 121
I micro e la medicina 126
Monitor o televisore? 132
Musica e computer 141
Le penne ottiche 156

I "micromondi" 164
Una penna... a motore 176
Immagini animate 181
Parola... al computer 186
I plotter 198
Pronti al decollo? 201
Affidarsi al caso 209
Gli accoppiatori acustici 216
I network 218
I "Floppy tape" 224

VOLUME SECONDO

Maghi del Flipper 241
I missili Cruise 243
Che tempo farà, domani? 248
Le tavolette grafiche 258
Una finestra sul mondo 264
I giochi... postali 266
I display a cristalli liquidi 278
Un servo obbediente 281
Uomini e topi 296
La comunicazione di massa 301
Il braccio e la mente 314
Il computer ergonomico 321
I joystick della nuova generazione 332
Viaggiare grazie ai computer 341
Il computer astronomo 346
I dischi Winchester 352
Inventiamo un gioco 361
Che accadrebbe se... 368
Le stampanti "a getto" 372
Fantascienza e realtà 381
I "sensi" del computer 394
Per i più piccoli 401
Il computer stenografo 414
Il Computer Aided Design 421
I dischi a laser 434
Guerra e pace 441
Il riconoscimento vocale 446
Il futuro 466

VOLUME TERZO

Cosa imparare per il futuro? 481
Software in vetrina 493
Il Wordstar 506
Software in vetrina 529
Software in vetrina 553
Word processing sul BBC! 554
L'incredibile Hulk 561
Software in vetrina 568
Facile accesso 581
Software in vetrina 588
Software in vetrina 609
Le immagini viewdata 612
Lo standard MSX 621
Software in vetrina 628
Sopravviva il migliore 641
Software in vetrina 648
I sistemi operativi 661
Software in vetrina 668
Solchi di luce 681
Software in vetrina 688
Tutto sotto controllo 701
Software in vetrina 715

VOLUME QUARTO

Il Basicode 721
Software in vetrina 728
Divoratori di parole 741
Software in vetrina 748
Database e computer 761
Software in vetrina 768
Scaccomatto al computer 781
Software in vetrina 788
Una rete di computer 801
Software in vetrina 808
I computer... in volo 821
Software in vetrina 828
Tecniche di marketing 841
Software in vetrina 848
Giocare via modem 864
Software in vetrina 868
I "business games" 881
Software in vetrina 888
Il mondo in un computer 901
Software in vetrina 908
Un bridgista d'eccezione 924
Software in vetrina 928
Il computer ci controlla? 941
Software in vetrina 948

VOLUME QUINTO

I creatori di melodie 961
Crimine al computer 966
Software in vetrina 968
Software in vetrina 988
Nuove espressioni musicali 989
Confezioni "su misura" 1001
Suoni in sequenza 1014
Le interfacce musicali 1033
Musica... sintetica 1041
Gran finale 1061
Paperi e droidi 1081
Un passo alla volta 1101
Topi e labirinti 1121
Mani e gomiti 1141
I "sensi" dei robot 1161
Movimenti di precisione 1181

VOLUME SESTO

Percorso a ostacoli 1201
Gli occhi del robot 1221
Due chiacchere col robot 1241
La conoscenza 1261
Simulare un robot 1281
Piccoli robot 1301
Bracci robotizzati 1321
I robot nel futuro 1341
Software di taglio europeo 1361
Telematica per tutti 1381
Comunicare tramite computer 1401
Software in vetrina 1420
Strumenti per comunicare 1421
Software in vetrina 1440

VOLUME SETTIMO

Un esempio da seguire? 1443
I Bulletin Board 1461
Il computer insegna 1481
Scrivere col computer 1501
Necessità particolari 1521
Viaggio nella didattica 1541
L'allievo comanda 1561
Una banca dati in classe 1581
Software in vetrina 1600
Linguaggi moderni 1601
Il Computer Assisted Learning 1621
Una lezione di economia 1641
Le classi del futuro 1661

VOLUME OTTAVO

La mente domina la materia 1681
Intelligenza e cervello 1701
Sistemi di ricerca 1721
Proiezioni strategiche 1748
Metodi Scout e B-Star 1761
Esperto a portata di chip 1781
Come e cosa insegnare ai computer 1801
Codici genetici 1828
Questioni di linguaggio 1841
Rappresentare la conoscenza 1843
Vedere e capire 1861
Riconoscere immagini 1881
Tra il dire e il fare... 1901

I pionieri dell'informatica

VOLUME PRIMO

Sir Clive Sinclair 120
John von Neumann 140
Steve Wozniak 155
Chuck Peddle 180
Alan Turing 200
Charles Babbage 220
Herman Hollerith 240

VOLUME SECONDO

Gottfried Leibniz 260
Norbert Wiener 300
Lyons: pasticcini ed informatica 320
Konrad Zuse 340
Leonardo Torres 360
Powers e Hollerith 380
Vannevar Bush 400
La Bell Laboratories 420

Grace Hopper 440
Il MARK 1 460
Tutti gli uomini del... computer 478

VOLUME TERZO

Bill Gates 500
Il successo della Atari 519
La Acorn Computers 540
Il software della Imagine 559
Entra in scena la Xeros 580
Le menti della Commodore 599
La Oric 620
La giapponese SORD 639
La Virgin Games 679
La Tandy Corporation 680
La Zilog 700
La Digital Research 719

VOLUME QUARTO

La Camputers 740
La Quicksilva 760
La Olivetti 780
La Dragon 800
La Bug-Byte 820
La Motorola 840
La Llamasoft 860
La Sharp 880
La Artic Computing 900
La Texas Instruments 920
La Audiogenic 940
La Prism 960

VOLUME QUINTO

La Softsel 980
La Psion 1000
La Melbourn House 1020
La Casio 1040
La Memotech 1060

Le chiavi dell'Informatica ?

VOLUME PRIMO

Bit e byte 28
Memoria di ferro 58
Vero o falso? 68
Le porte logiche e i sommatori 92
I tipi di memoria 96
Il dialogo digitale 112
Le leggi del pensiero 128
Il cervello di un computer 138
La CPU e gli indirizzi 144
Il codice esadecimale 179
PEEK e POKE 188
Le interfacce 206
Cosa sono i "buffer"? 236

VOLUME SECONDO

Il controllo di parità 253
La generazione di caratteri 269
Il codice Hamming 298
Un redattore ineccepibile 308
Tracce e settori 324
Un passo alla volta 348
Le mappe di memoria 364
I generatori di programmi 384
I chip ULA 388
Un nuovo sort 413
La macchina di Turing 424
Il Computerese 428
Il linguaggio macchina 448
Qualche passo in Assembly 464
Salti generazionali 468

VOLUME TERZO

L'algebra di Boole 488
Sommare... con logica 512
Processi logici semplificati 526
La logica del programmatore 546
Le mappe di Karnaugh 572
Dalla mappa al circuito 586
Una risposta logica 606
La conversione dei segnali 626
Strade alternative 646
Un po' di parità 666
Un convertitore per display 686
I flip-flop RS 708

VOLUME QUARTO

Circuiti temporizzatori 726
A spasso per la memoria 746
L'unità logica 772

Programmazione BASIC >

VOLUME PRIMO

Obbedire... e basta! 20
Gira e rigira 38
I puntini sulle i 52
Parliamo di routine 77
Quanto manca a Natale? 98
Le variabili indicizzate 116
La struttura di un programma 134
Le funzioni intrinseche 146
Un numero... a caso 172
Le variabili a due dimensioni 194
Facciamo il punto 212
Una rubrica telefonica... computerizzata 232

VOLUME SECONDO

Scegliere la struttura adatta 254
L'inserimento di nuovi nominativi 272
Soluzioni degli esercizi di riepilogo 280
Scegliamo il menu adatto! 292
Come usare i file 316
Scambi e ricerche 336
Dai moduli al programma intero 354
Un "giro" di prova 376
Velocità di ricerca 396
Chi cerca... 416
Ritocchi finali 436
Una questione di stile 456
Quanto vale il BASIC? 474

VOLUME TERZO

Il Basic dello Spectrum 494
Funzioni e strutture di controllo 504
L'ABC del... BBC! 534
Il BASIC dei Commodore 574
Qualche grado... in BASIC 634
Gradi di precisione 654
La grafica del Commodore 694
Disegniamo al computer 704
Gli sprite del Commodore 712

VOLUME QUARTO

Che bestia è? 732
Alta e bassa risoluzione 734
Sottomarini ed esplosioni 744
I suoni del computer 754
Sprite... subacquei 764
Il quadrato magico 766
Carovana nel deserto 792
Via ai sottomarini! 794
Un mare di onde 812
Spectrum in rete 826
Atterraggio lunare 832
La tastiera dello Spectrum 846
Colori e numeri 852
Grafica BBC ed Electron 872
Logica divertente 879
Campo visuale 884
I programmi diagnostici 892
Il Sabre Wulf 913
Rotta di collisione 914
Prontezza di riflessi 917
Scoppi e tuoni 926
Cubi in rotazione 932
Spruzzi di colore 946
Percorsi circolari 954

VOLUME QUINTO

Fine alle mine 972
Il gioco dell'impiccato 984
Un generatore di caratteri 1052
La creazione di caratteri 1068
Utili puntatori 1096
Una gara... in moto 1112
Un altro giro in moto 1133
Le routine di utilità 1144

Una utility per il Commodore 64 1180
Utility ibride 1184

VOLUME SESTO

Ricerca e sotituzione 1206
Variabili variate 1224
Spirito d'avventura 1246
Digitaya: i messaggi 1272
Digitaya: gli ambienti 1293
Accettare ordini 1306
Raccogliere oggetti 1326
Abbandonare oggetti 1346
Spiriti e fantasmi 1363
Avventura grafica 1384
Più schermi sullo Spectrum 1412
Più schermi sul CBM 64 1432

VOLUME SETTIMO

Nuotare o annegare 1446
La Foresta Incantata 1477
Digitaya 1497
Un gioco di simulazione 1506
Tutti a bordo! 1532
Alimenti per la ciurma 1546
Ultimi acquisti 1566
Pronti a salpare 1586
In alto mare 1612
Uomo in mare! 1624
Navigatori solitari 1652
In un mare di guai 1672

VOLUME OTTAVO

Serrare i portelli di boccaporto 1692
Ammutinamento a bordo 1712
Approdo al Nuovo Mondo 1724
Ratto baratto 1746
Profitti e perdite 1774
Nuovo Mondo I 1794
Nuovo Mondo II 1818
Nuovo Mondo III 1834

Note d'informatica

VOLUME QUINTO

Introduzione al LOGO 986
LOGO e tartarughe 1012
Divertirsi con la tartaruga 1023
Giochi d'incastro 1044
Variazione su... una tartaruga 1073
La ricursione del LOGO 1084
"Divide et impera" 1103
Una tartaruga spaziale 1134
Sprite col LOGO 1146
Caccia al drago 1174
Il LOGO per gli Atari 1186

VOLUME SESTO

Lavorare sui numeri 1215
Un po' di poesia 1234
Giocare in LOGO 1254
Di luogo in... LOGO 1266
Fine di un'avventura 1288
Chi è stato? 1312
Procedure autogeneranti 1333
Grafici in LOGO 1353
Disegnare cicloidi 1366
Trasformazioni spaziali 1395
Altre trasformazioni 1414
Fattoriali! 1434

VOLUME SETTIMO

Traguardo finale 1452
Un linguaggio disciplinato 1466
Prime parole 1484
Tipi di variabili 1504
Decisioni costruttive 1526
1 + 2 = 20? 1554
Ripetila ancora RAM! 1572
Gli insiemi nel PASCAL 1594
A tempo di... record 1614
Le matrici in PASCAL 1626
Linguistica strutturale 1658
Variabili di valore 1664

VOLUME OTTAVO

Strutture dinamiche 1684
Un duetto dinamico 1714
GOTO END 1726
Un prologo sul PROLOG 1752
Occhiate inquisitrici 1792
Effetti collaterali 1806
Analisi introspettiva 1832
Il mondo delle liste 1846
Liste e funzioni in LISP 1864
Liste attive 1886
Il FORTH 1908

Linguaggio macchina

VOLUME TERZO

Gli aspetti di base 496
I banchi di memoria 516
I programmi monitor 536
Analisi del testo 556
Alcune utili convenzioni 576
Byte per byte 596
Largo alla memoria 615
Linea di partenza 636
Sventola una bandiera 656
Modi d'indirizzamento 676
Etichette e cicli 696
Subroutine trasportabili 716

VOLUME QUARTO

Last In First Out 737
Somme e sottrazioni 756
Moltiplicazioni e rotazioni 777
Le divisioni 796
Pixel su pixel 817
Gli sprite sullo Spectrum 837
Sprite sul BBC 857
Finestre sullo Spectrum 876
Piani inclinati 896
Un cerchio di luce 918
I cerchi sul Commodore 937
Un attimo di tregua 958

VOLUME QUINTO

Colorare col Commodore 976
L'Assembly del 6809 998
Da registro a registro 1017
Database del Chip 6809 1029
I campi del 6809 1038
Database del Chip 6809 1056
L'Aritmetica del 6809 1057
Database per Z80 della Zilog 1072
Cambio d'indirizzo 1078
Database per Z80 della Zilog 1088
Ricerca e abbinamento 1098
Database per Z80 della Zilog 1108
Salti ad effetto 1117
Database per Z80 della Zilog 1129
Uso degli stack 1137
Database per Z80 della Zilog 1156
Ingresso e uscita dal 6809 1157
Database per Z80 della Zilog 1168
Sospendere l'esecuzione 1177
Database per Z80 della Zilog 1196
Libero movimento 1197

VOLUME SESTO

Database per Z80 della Zilog 1208
Una struttura sensata 1218
Database per Z80 della Zilog 1237
Un utile debugger 1238
Database per Z80 della Zilog 1249
Bug sotto controllo 1257
Database istruzioni per il 6502 1274
Un punto d'arresto 1275
Database istruzioni per il 6502 1296
Ultimi comandi 1297
Database caratteri sostitutivi nazionali 1309
Ritocchi finali 1318
Database per routine ROM del Micro BBC 1332
Concetti operativi 1338
Database per routine ROM del Micro BBC 1352
Il sistema operativo del BBC 1358
Uso di vettori nel BBC 1377

Database per routine ROM del Micro BBC 1388
Schermi a volontà 1398
Database per routine ROM del Micro BBC 1408
Movimenti scorrevoli 1417
Database per routine ROM del Micro BBC 1428
Le chiamate OSBYTE del BBC 1437

VOLUME SETTIMO

Database per routine ROM del Micro BBC 1449
Le chiamate OSWORD del BBC 1458
Database per routine ROM del Micro BBC 1468
La gestione dei file sul BBC 1474
Database per routine ROM del Micro BBC 1488
Ancora sui file del BBC 1495
Database per routine ROM del Micro BBC 1508
Il nuovo e il vecchio 1517
Database per routine ROM del Micro BBC 1529
Interazioni del BASIC 1536
Database per routine ROM del Micro BBC 1548
Interrompiamo il programma... 1557
Databse per routine ROM del Micro BBC 1569
Gli eventi principali 1579
Ultima chiamata 1597
Database per mappa di memoria del Commodore 64 1608
Il SO del Commodore 64 1617
Database per mappa di memoria del Commodore 64 1629
I/O sul Commodore 64 1637
Linee di comunicazione 1646
Database per mappa di memoria del Commodore 64 1648
Database per mappa di memoria del Commodore 64 1668
Effetti ottici 1678

VOLUME OTTAVO

Database per mappa di memoria del Commodore 64 1695
Schermi multipli 1698
Database per mappa di memoria del Commodore 64 1708
Il "beat" dei byte 1717
Database per mappa di memoria del Commodore 64 1732
Disegni reticolari 1736
Rivoluzione grafica 1756
Database per mappa di memoria del Commodore 64 1760
Database per mappa di memoria del Commodore 64 1769
Dichiarazioni d'interesse 1772
I segreti dello Spectrum 1776
Database per mappa di memoria del Commodore 64 1788
Canali di pensiero 1797
Flussi di controllo 1812
Database per mappa di memoria del Commodore 64 1820
Nastri e misure 1837
Interrupt e utili routine 1856
Microdrive e porta seriale 1876
Reti LAN e nuovi commandi BASIC 1897
Ultime routine ROM 1910

Laboratorio

VOLUME TERZO

I primi passi 524
Dissaldare 548
Banco di prova 566
Le leggi elettriche 594
L'interno dei componenti 618
Contatti... logici 624
Mezza misura 644
Facciamo il punto 674
Soluzioni agli esercizi 689

VOLUME QUINTO

Una porta... da aprire 994
Una porta per comandare 1003
Collaudiamo la "scatola nera" 1026
Dispositivi di controllo 1054
Controllo di un motore 1065
Duplice controllo 1092
Controllo del traffico 1114
Fonte d'energia 1126
Semplici applicazioni 1154
Un comodo visualizzatore 1165
Un convertitore D-A 1194

VOLUME SESTO

Produrre suoni 1212
Una risposta... a tono 1226
Perso nel labirinto 1252
Tiriamo le somme 1278
Un semplice robot 1290
Motore fuoribordo 1315
Riuniamo le parti 1336
Micro elettronica 1356
Taratura del robot 1374
I servomotori 1392
Usare i servomotori 1403
Programmi per il robot 1423

VOLUME SETTIMO

Missione di ricognizione 1455
Interfaccia Spectrum 1472
Realizziamo l'interfaccia 1492
Collegare l'interfaccia 1514
A me gli occhi! 1538
Una sofisticata "mano" 1552
Braccio di ferro 1574
Montiamo il braccio 1592
Muscoli e presa 1606
Fine di un braccio 1630
A portata di mano 1655
Olio di gomito 1675

VOLUME OTTAVO

I tendini del Sinclair 1696
La curvatura del braccio 1706
Un tracciatore digitale 1733
Montaggio del Tracer 1754
Tracce per la memoria 1766
Tracciamenti fantasiosi 1786
Un'interfaccia MIDI 1815
Circuiti audio 1823
Linee di melodia 1848
Regole di composizione 1866
Hi-Fi con la MIDI 1892

Tecniche di programmazione

VOLUME QUARTO

I trucchi del mestiere 814
Ci vuole stile... 834
Piano d'azione 854
Gli algoritmi 866
Cicli e iterazioni 894
Mosse decisive 904
Programmazione modulare 934
Dai concetti alla pratica 956

VOLUME QUINTO

Errori? No, grazie 964
L'elemento umano 992
I menu di assistenza 1007
La scelta... di una scelta 1036
Le "librerie" di routine 1046
Software calibrato 1076
Il check-up di un programma 1086

Suoni e luci

VOLUME SECONDO

Effetti speciali 246
Consigli sul suono... e sulla grafica 276
I suoni del Vic... le luci del Dragon 284
Semplici suoni... Immagini primarie 312
Sistemi audio... Programmi luminosi 334
Qualità musicali... Lampi di luce 358
A prova di suono / Immagini d'autore 374
Suoni spettacolari / Immagini alla ribalta 392
Suoni incredibili / Luci sfolgoranti 408
Idee sonore / Onde di luce 426
Principi del suono / Luci rinfrescanti 452
Suoni ideali / Luci di posizione 472

21498