Spediz. in abbonamento postale GR. II/70 L. 2.000
(...)

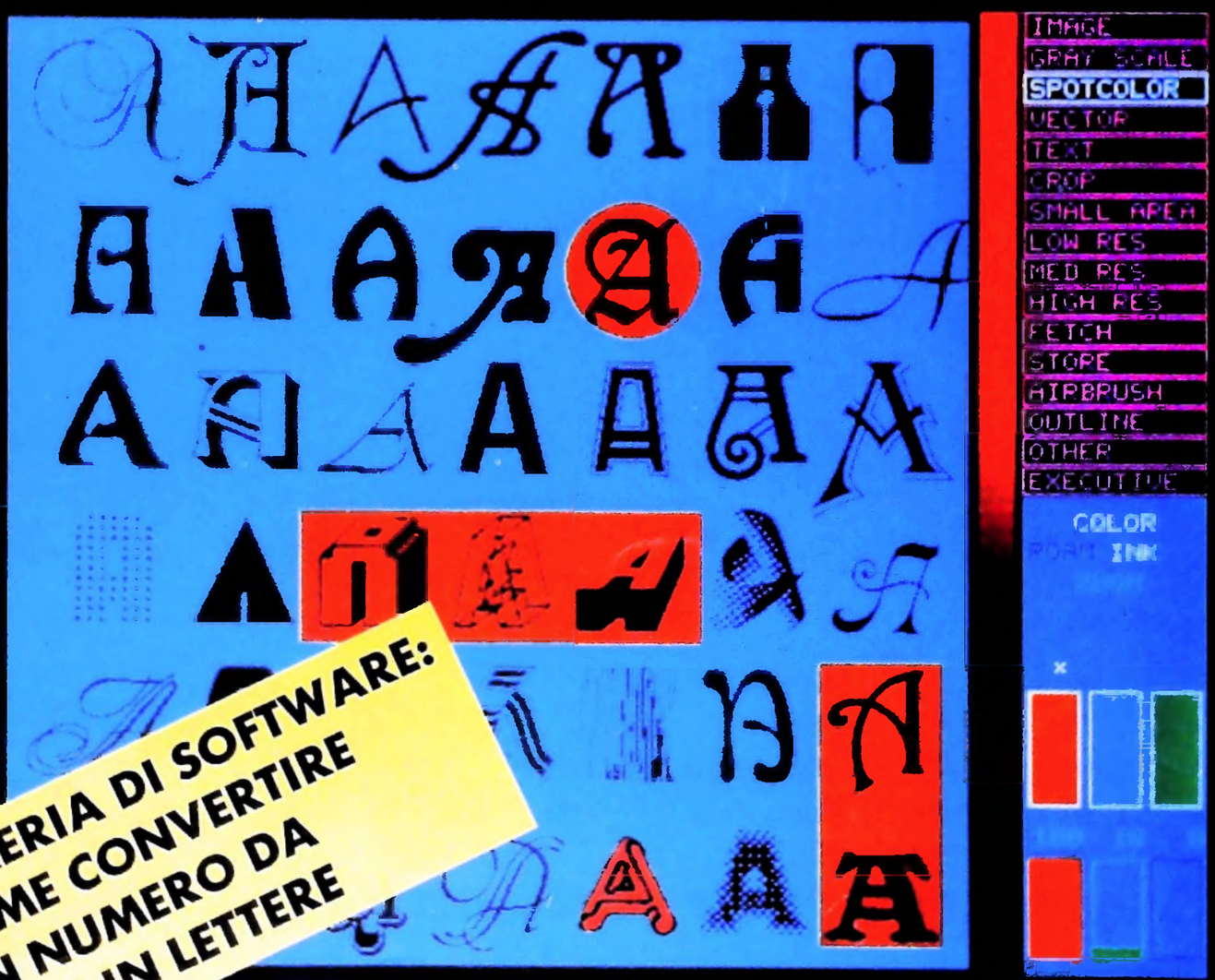
16 CORSO PRATICO COL COMPUTER

421685

è una iniziativa
FABBRI EDITORI
in collaborazione con
BANCO DI ROMA
e **OLIVETTI**

F4 F5 F6 F7 F8 BATTERY LOW

diretto da GIANNI DEGLI ANTONI



LIBRERIA DI SOFTWARE:
COME CONVERTIRE UN NUMERO DA CIFRE IN LETTERE

FABBRI EDITORI

È in edicola il sesto numero di LIBRERIA DI SOFTWARE dedicato a Programmi di statistica - 1 PARAM: valutazione dei parametri statistici

Direttore dell'opera
GIANNI DEGLI ANTONI

Comitato Scientifico
GIANNI DEGLI ANTONI
Docente di Teoria dell'informazione, Direttore dell'Istituto di Cibernetica dell'Università degli Studi di Milano

UMBERTO ECO
Ordinario di Semiotica presso l'Università di Bologna

MARIO ITALIANI
Ordinario di Teoria e Applicazione delle Macchine Calcolatrici presso l'Istituto di Cibernetica dell'Università degli Studi di Milano

MARCO MAIOCCHI
Professore Incaricato di Teoria e Applicazione delle Macchine Calcolatrici presso l'Istituto di Cibernetica dell'Università degli Studi di Milano

DANIELE MARINI
Ricercatore universitario presso l'Istituto di Cibernetica dell'Università degli Studi di Milano

Curatori di rubriche
TULLIO CHERSI, ADRIANO DE LUCA (Professore di Architettura dei Calcolatori all'Università Autonoma Metropolitana di Città del Messico), GOFFREDO HAUS, MARCO MAIOCCHI, DANIELE MARINI, GIANCARLO MAURI, CLAUDIO PARMELLI, ENNIO PROVERA

Testi
ADRIANO DE LUCA, CLAUDIO PARMELLI, LUCA SPAMPINATO, Etnoteam (ADRIANA BICEGO)

Tavole
Logical Studio Communication
Il Corso di Programmazione e BASIC è stato realizzato da Etnoteam S.p.A., Milano
Computergrafica è stato realizzato da Eidos, S.c.r.l., Milano
Usare il Computer è stato realizzato in collaborazione con PARSEC S.N.C. - Milano

Direttore Editoriale
ORSOLA FENGHI

Coordinatore settore scientifico
UGO SCAIONI

Redazione
MARINA GIORGETTI
LOGICAL STUDIO COMMUNICATION

Art Director
CESARE BARONI

Impaginazione
BRUNO DE CHECCHI
PAOLA ROZZA

Programmazione Editoriale
ROSANNA ZERBARINI
GIOVANNA BREGGÉ

Segretarie di Redazione
RENATA FRIGOLI
LUCIA MONTANARI

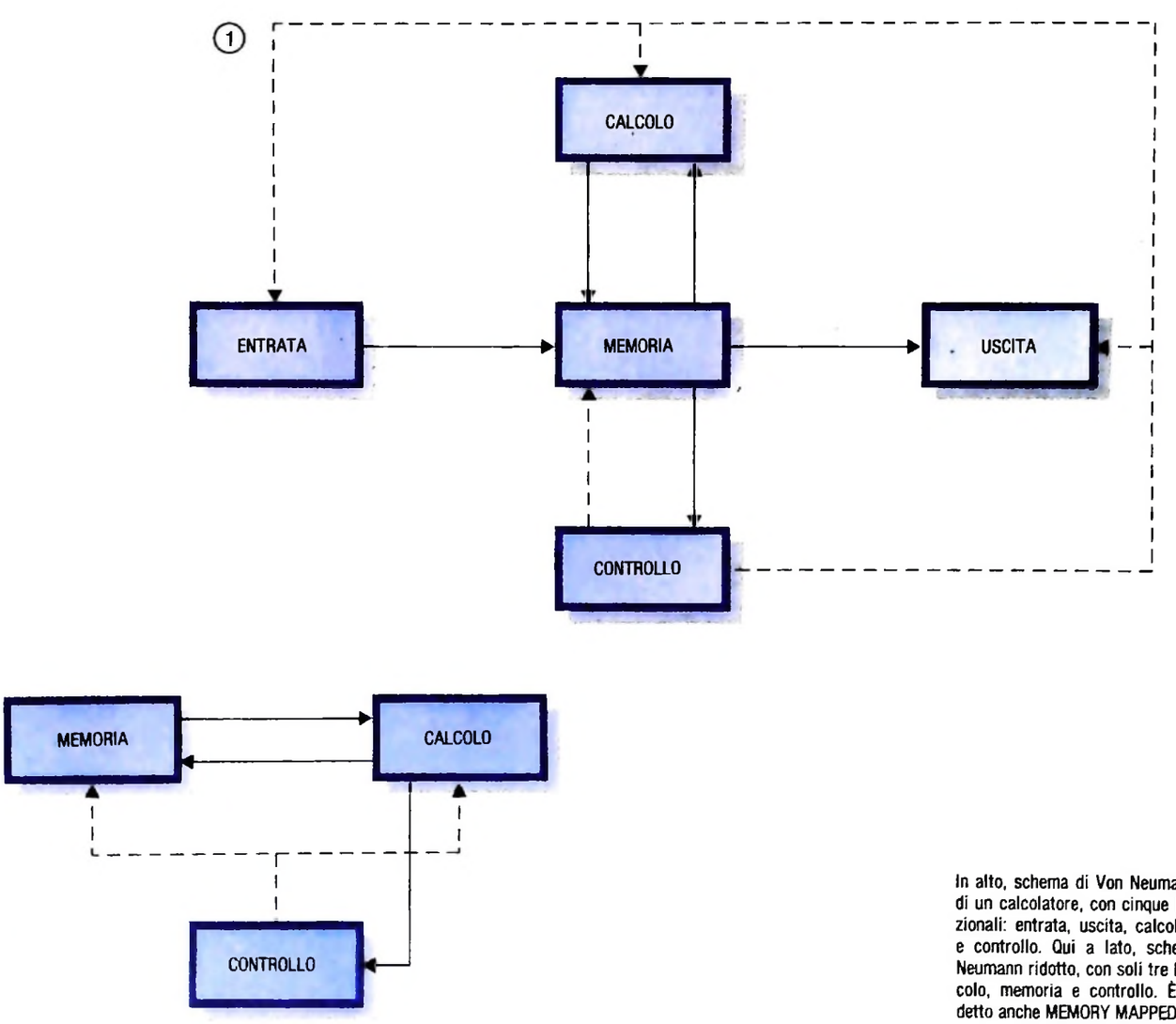
NEL PROSSIMO NUMERO IN OMAGGIO IL SETTIMO POSTER "LA STORIA DELL'INFORMATICA"

Corso Pratico col Computer - Copyright © sul fascicolo 1984 Gruppo Editoriale Fabbri, Bompiani, Sonzogno, Etas S.p.A., Milano - Copyright © sull'opera 1984 Gruppo Editoriale Fabbri, Bompiani, Sonzogno, Etas S.p.A., Milano - Prima Edizione 1984 - Direttore responsabile GIOVANNI GIOVANNINI - Registrazione presso il Tribunale di Milano n. 135 del 10 marzo 1984 - Iscrizione al Registro Nazionale della Stampa n. 00262, vol. 3, Foglio 489 del 20.9.1982 - Stampato presso lo Stabilimento Grafico del Gruppo Editoriale Fabbri S.p.A., Milano - Diffusione Gruppo Editoriale Fabbri S.p.A. via Mecenate, 91 - tel. 50951 - Milano - Distribuzione per l'Italia: A. & G. Marco s.a.s., via Fortezza 27 - tel. 2526 - Milano - Pubblicazione periodica settimanale - Anno I - n. 16 - esce il giovedì - Spedizione in abb. postale - Gruppo II/70. L'Editore si riserva la facoltà di modificare il prezzo nel corso della pubblicazione, se costretto da mutate condizioni di mercato.

L'UAMICRO

Una famiglia di microprocessori concepita per uso didattico e capace di applicazioni anche industriali.

I microprocessori della famiglia UAMICRO, di cui inizia adesso la descrizione, sono microprocessori didattici progettati e costruiti in laboratori universitari. Essi hanno tutte le caratteristiche dei microprocessori commerciali di tipo dedicato (DEDICATED), cioè microprocessori a funzioni specifiche. Essi sono stati realizzati allo scopo di facilitare la comprensione dei meccanismi di funzionamento tanto a livello interno come a livello esterno.
La filosofia su cui fu basato lo sviluppo di questi microprocessori è simile a quella adottata per tutti gli altri, cioè lo schema generale di Von Neumann (figura 1). Tuttavia, l'architettura dell'UAMICRO è basata sullo schema di Von Neumann ridotto. La differenza fra i due schemi non consiste soltanto nell'integrazione di alcuni blocchi funzionali, cioè la riduzione dei tre blocchi ENTRATA, MEMORIA, USCITA a uno solo, MEMORIA, quanto piuttosto nelle condizioni specifiche di funzionamento che quest'ultima configurazione comporta.



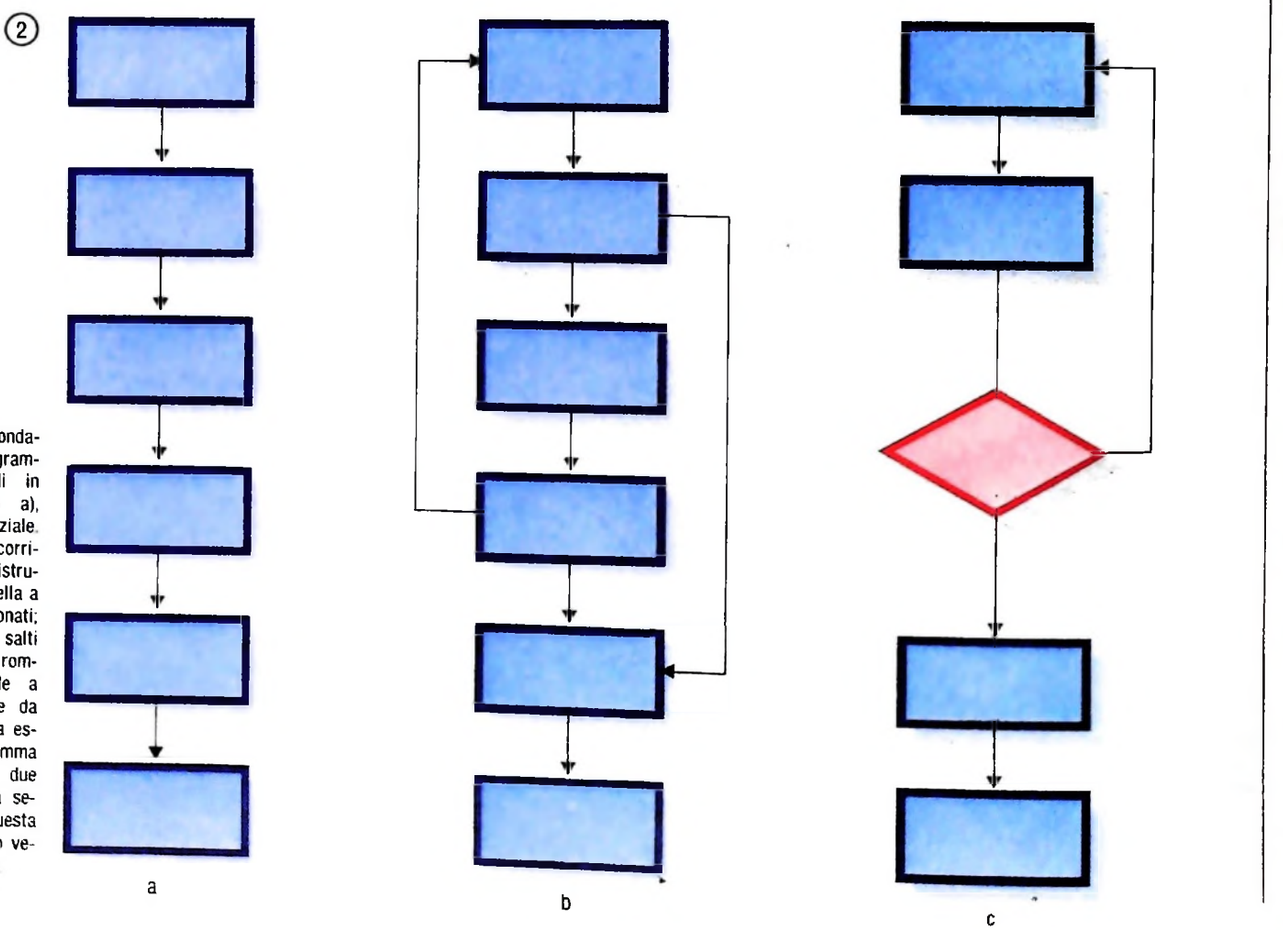
In alto, schema di Von Neumann classico di un calcolatore, con cinque blocchi funzionali: entrata, uscita, calcolo, memoria e controllo. Qui a lato, schema di Von Neumann ridotto, con soli tre blocchi: calcolo, memoria e controllo. È lo schema detto anche MEMORY MAPPED.

Quando parliamo dello schema di Von Neumann ridotto, in realtà ci stiamo riferendo allo schema chiamato comunemente di MEMORY MAPPED, cioè quello schema che considera tutti gli elementi esterni alla CPU come dei generici registri. Questi registri esterni, come vedremo, possono essere registri di memoria vera e propria, registri di controllo o registri di entrata e uscita. Il protocollo che la CPU usa per leggere o scrivere su uno qualunque di essi, però, è sempre lo stesso. L'unica differenza sta nel tempo necessario per la lettura o la scrittura, che può variare secondo il tipo di registro. A monte ancora di questa filosofia c'è anche il fatto importante che riguarda l'insieme delle istruzioni di un microprocessore; nel primo sistema (Von Neumann completo) esistono delle istruzioni specifiche per i registri di entrata e uscita diverse dalle istruzioni che riguardano la memoria. Ciò vuol dire anche che esiste una configurazione speciale di interfaccia con i registri di entrata e uscita che ne facilita il collegamento, anche se poi un'interfaccia simile riduce il numero di registri possibili da collegare.

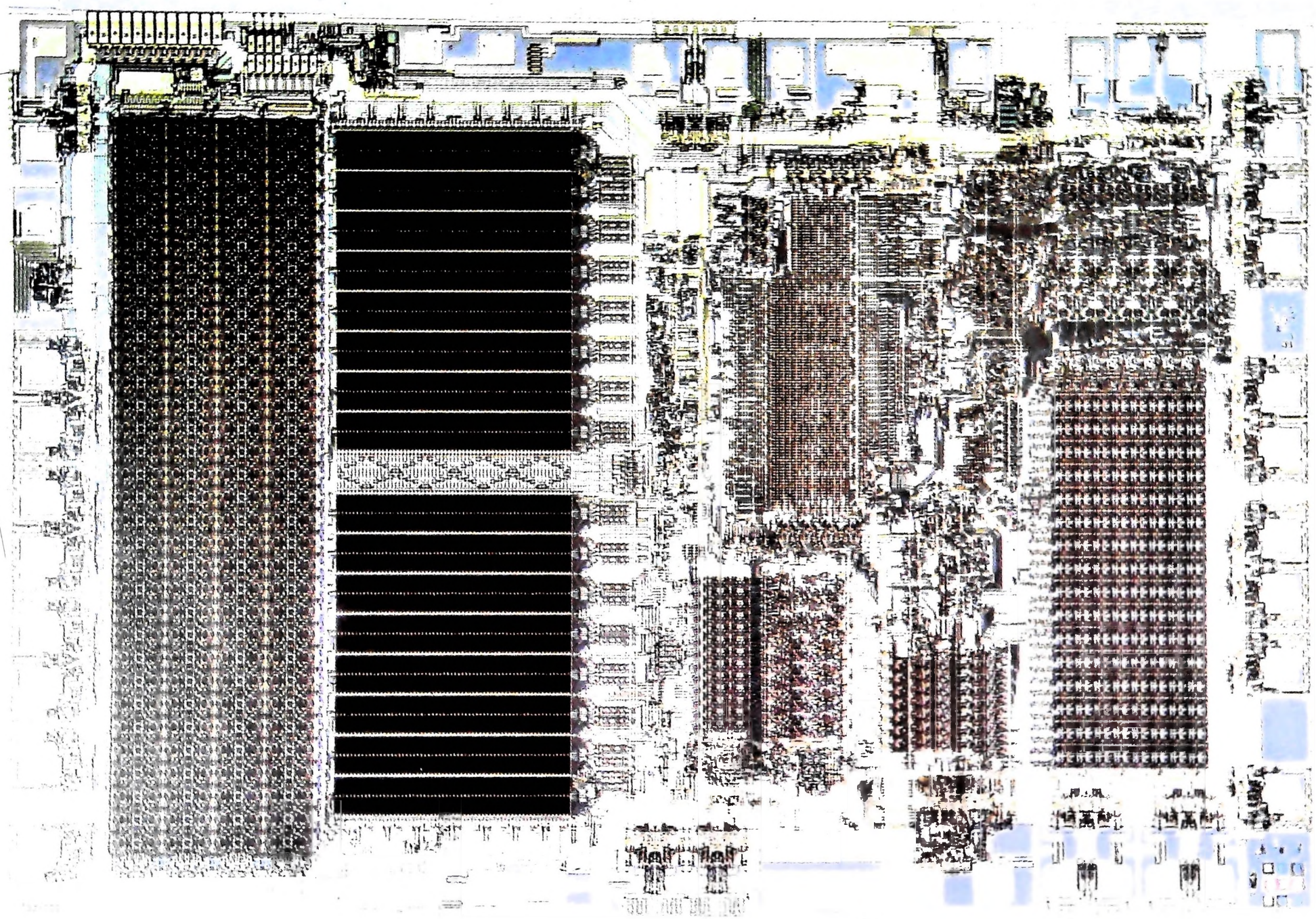
Un esempio di questa architettura è lo Z80 che permette di collegare con facilità un massimo di 256 registri di ENTRATA e USCITA, quantità che in pratica risulta anche eccessiva. Per ciò che riguarda invece il sistema MEMORY MAPPED, abbiamo la possibilità di collegare un numero di registri di ENTRATA e USCITA che, al limite, può essere uguale al numero di locazioni di memoria disponibile e, ciò che è più importante, tutte le istruzioni di memoria vera e propria possono essere dedicate ai registri di ENTRATA e USCITA. I dispositivi in commercio adoperano ambedue queste tecniche: lo schema di Von Neumann completo è usato da Intel, Zilog e altri, mentre lo schema MEMORY MAPPED è usato da Motorola e altri. Da notare come lo schema di Von Neumann ridotto sia formato da soli tre blocchi.

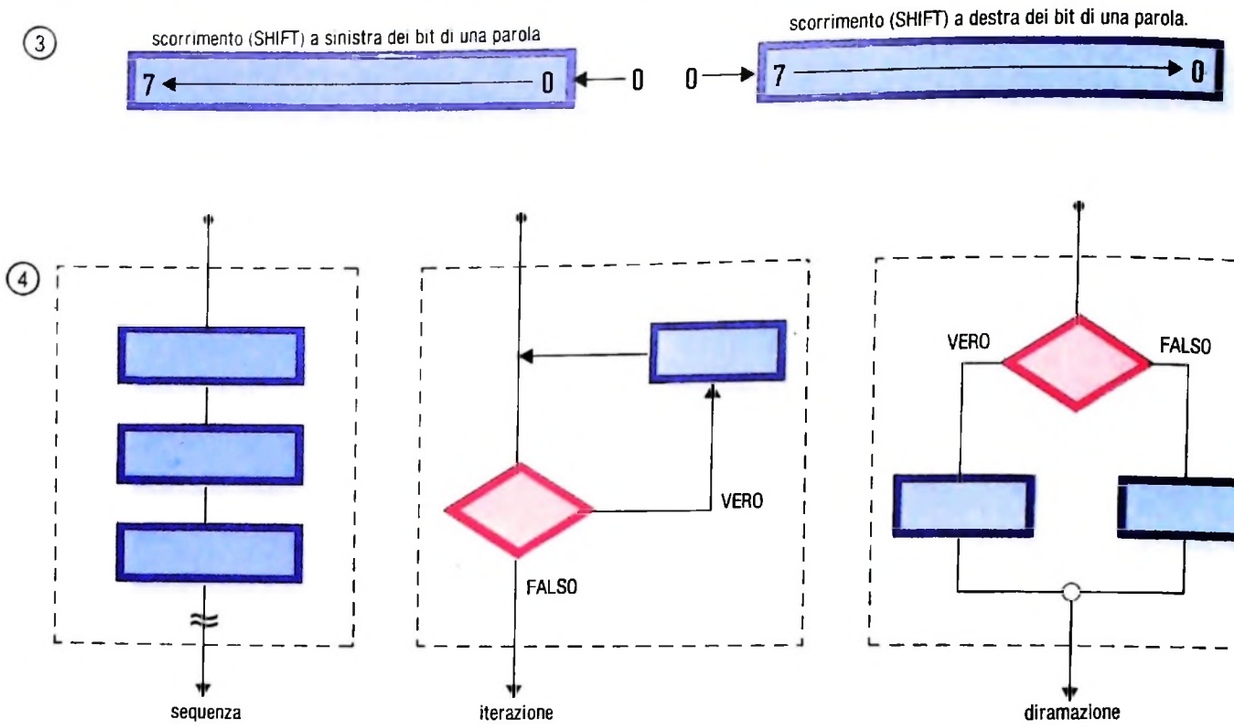
Il microprocessore a schema di Von Neumann

Qui a destra un microprocessore a schema di Von Neumann completo (38SH72 SGS). È un microcalcolatore (*computer on a chip*), dotato di una memoria RAM propria di caratteristiche avanzate. Si tratta infatti di una memoria non volatile (che conserva cioè il proprio contenuto di informazioni anche quando il microcalcolatore non è alimentato da una corrente). Questo tipo di RAM è detta SHRAM, cioè Secondary Hold RAM (RAM a immagazzinamento secondario). È in pratica una RAM dinamica connessa a una memoria non volatile: se la corrente che alimenta la RAM dinamica cade sotto i 4,5V, questa scarica il suo contenuto in un elemento di memoria non volatile; rialimentando il circuito, la RAM dinamica recupera le informazioni dall'elemento di memoria non volatile.



Tre strutture fondamentali di programmi realizzabili in hardware. In a), quella sequenziale. Ogni blocco corrisponde a una istruzione; in b) quella a salti incondizionati; in c), quella a salti condizionati. Il rombo corrisponde a una condizione da verificare, e da esso il programma prosegue in due modi diversi a seconda che questa dia un risultato vero oppure falso





Tre blocchi fondamentali nella costruzione di un programma strutturato. Quello al centro (iterazione) corrisponde a FOR, oppure al WHILE ... DO di molti linguaggi di alto livello. Quello a destra (diramazione) corrisponde a IF ... THEN ... ELSE.

Le strutture di programmi dell'UAMICRO

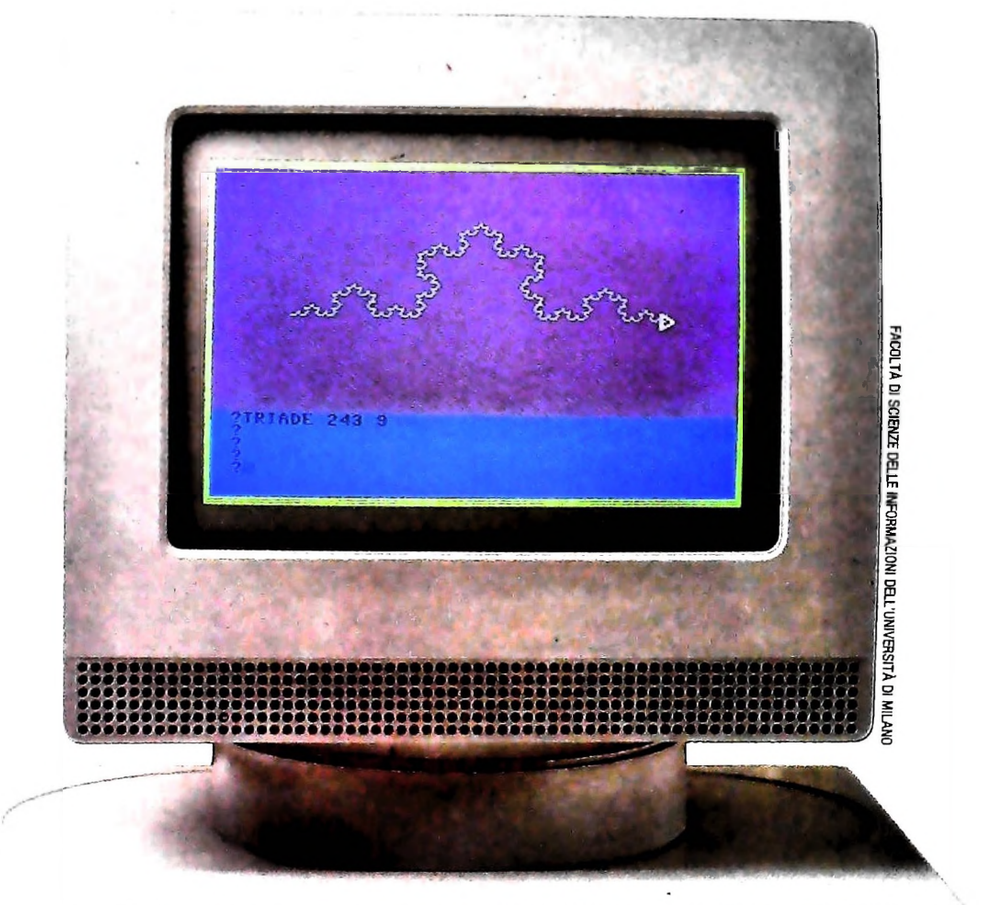
Una delle caratteristiche principali dell'architettura dei microprocessori UAMICRO, comune a tutti i microprocessori, è la possibilità di realizzare le strutture di programma mostrate in figura 2:
a) sequenziale
b) a salti incondizionati
c) a salti condizionati.
La scrittura di sequenza (a) è quella in base alla quale le istruzioni vengono eseguite l'una dopo l'altra in forma automatica. La struttura a salti condizionati (c) è quella in cui da un'istruzione di tipo condizionato (rappresentata con un rombo), a seconda della condizione, si passa alla successiva oppure si salta indietro o in avanti di alcune istruzioni. Nel caso b, invece, salti incondizionati, quando l'esecuzione delle istruzioni arriva a un punto determinato l'esecuzione salta indietro o avanti di alcune istruzioni.
La vera differenza che esiste fra la struttura di salto condizionato e quella di salto incondizionato è che la condizione di salto nella prima è data al momento del salto, mentre nella seconda è data prima o dopo il verificarsi del salto.

Le funzioni dell'UAMICRO

Le altre funzioni realizzabili con l'UAMICRO sono:
a) funzioni logiche
b) funzioni aritmetiche
c) funzioni di entrata e uscita.
Vediamo ciò che vogliono dire. Le funzioni logiche consistono nella realizzazione di funzioni AND, OR, OR ESCLUSIVO, INVERSIONE, SET e RESET fra due parole e nella singola parola, secondo l'istruzione. Le funzioni aritmetiche sono la somma di due numeri, la sottrazione, il muovere una parola a destra o a sinistra (SHIFT) come mostra la figura 3, il che permette in questa forma la moltiplicazione o la divisione per due del valore numerico della parola.
Fermiamoci un istante a commentare l'importanza delle tre strutture di programma viste in figura 2. Oggi, dopo un passato abbastanza prolifico ma confuso per ciò che riguarda i linguaggi di programmazione, si è trovato il modo di rendere i programmi leggibili e anche modificabili da persone che non ne sono gli autori. Si è trovata cioè una forma di stesura del programma che, anche se occupa un po' più di spazio di memoria, in cambio è più facile da capire e trasformare. I linguaggi basati su questa nuova tecnica sono chiamati linguaggi "strutturati". Le tre strutture fondamentali in base alle quali è possibile realizzare un qualsiasi programma oppure creare delle altre strutture che lo rendono più maneggevole ed efficiente sono elencate in figura 4. Da un attento esame di queste strutture software rispetto alle strutture hardware interne al microprocessore si può dedurre quanto segue: per ciò che riguarda la struttura software sequenziale, questa è facilmente realizzabile con la struttura sequenziale interna all'hardware. Le strutture software di iterazione e di diramazione sono realizzabili con le strutture di salto incondizionato e di salto condizionato.

I LINGUAGGI DELL'INTELLIGENZA ARTIFICIALE

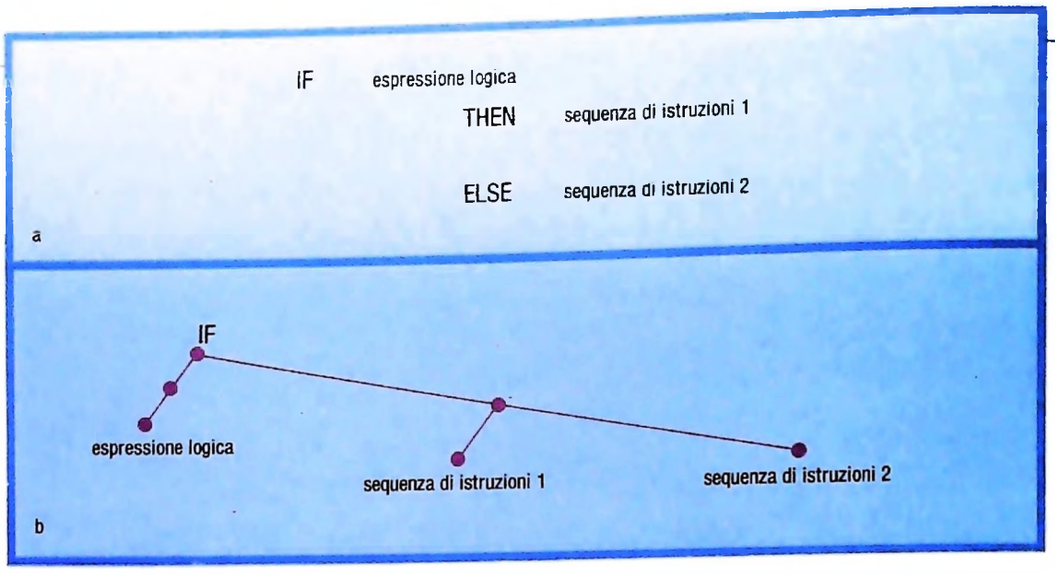
Le ricerche sull'intelligenza artificiale portano alla creazione di una gerarchia strutturata di linguaggi al cui culmine è quello "naturale".



FACOLTÀ DI SCIENZE DELLE INFORMAZIONI DELL'UNIVERSITÀ DI MILANO

Un esempio dell'impiego di un linguaggio evoluto (il LOGO) che permette di creare figure sullo schermo a colori e di porle in memoria per elaborazioni successive. La figura è un esempio di uno stadio di produzione di curve a dimensione frazionaria (i "frattali" di B. Mandelbrot).

In questo capitolo si cercherà di individuare e descrivere quegli aspetti dell'attività di produrre programmi che sono stati maggiormente influenzati dalla ricerca nel campo della Intelligenza Artificiale (A.I.).
Si è visto come la natura particolarmente complessa dei problemi considerati fece in modo che i ricercatori A.I. si trovassero ad esplorare nuove tecniche di indagine e di rappresentazione e gestione delle conoscenze acquisite. Si è anche accennato a come le difficoltà si propagassero fino alla fase di sviluppo dei programmi necessari alla sperimentazione. Ci si doveva cimentare con ostacoli che non erano mai stati incontrati nell'ambito di sviluppo di programmi più usuali come quelli per il calcolo scientifico o per le applicazioni gestionali. Si trattava in particolare di dover sviluppare programmi a partire da specifiche formalizzate solo parzialmente, in cui la difficoltà era principalmente esprimere in forma di programma idee e concetti piuttosto astratti e specifici del problema considerato.
In questi casi non è richiesta di norma una particolare efficienza. ma le dimensioni del codice prodotto rendono diffici-



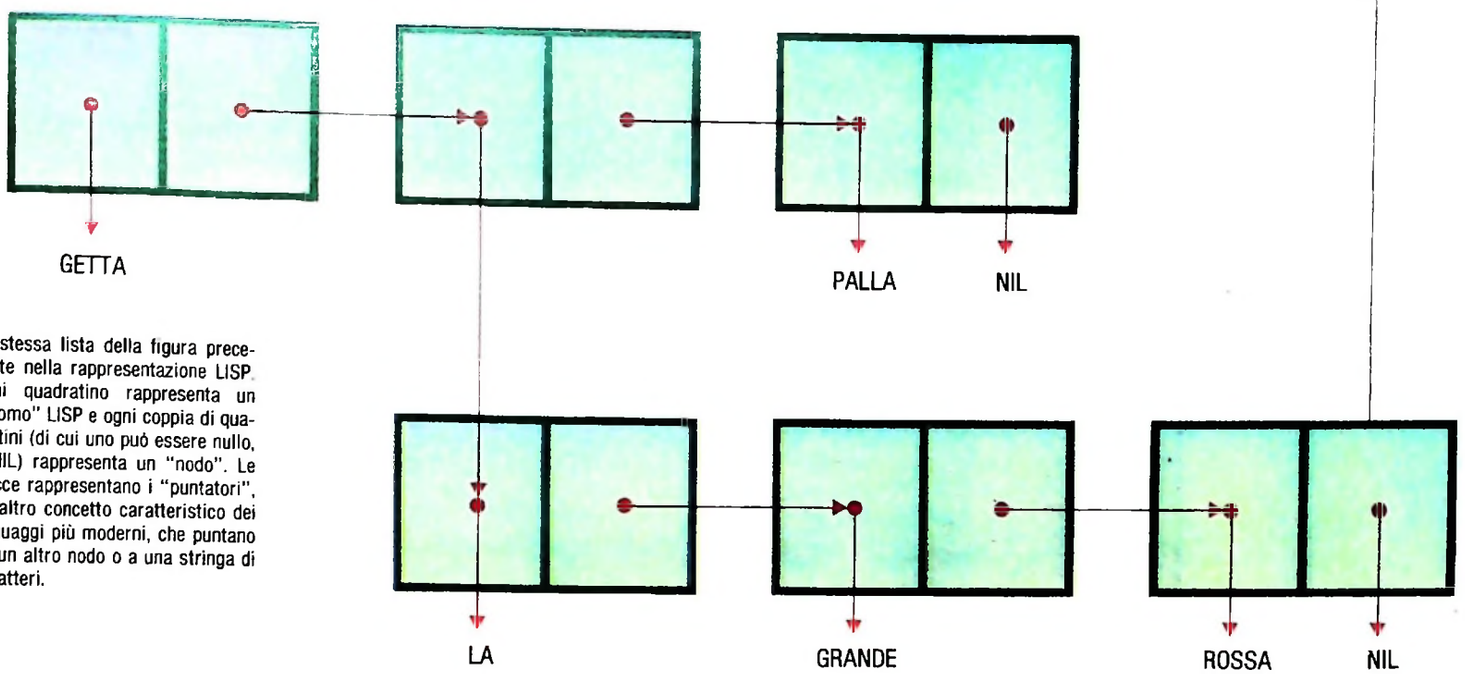
Nella rappresentazione strutturale (b) le parole chiave BASIC, THEN e ELSE, non sono utilizzate, mentre nella rappresentazione a caratteri (a) delimitano le due sequenze di istruzioni.

le controllare l'evoluzione del programma, specialmente se è necessaria una collaborazione tra diversi ricercatori. Come è stato più volte ricordato, il problema maggiore peraltro è che il programma prodotto deve continuamente confrontarsi con il modello che lo ha ispirato, il quale è soggetto a continue revisioni. Ne consegue che il programma stesso deve essere modificato molto spesso e quasi sempre in modo sostanziale. Inoltre è molto gradito che durante lo sviluppo di un grande sistema di A.I. le parti di codice prodotto possano essere in grado di funzionare prima che sia stato terminato lo sviluppo dell'intero programma. Questo implica che è necessario disporre di ambienti in cui sia "facile" scrivere e verificare programmi. Il primo sostanziale intervento dell'A.I. sul modo di produrre software si ebbe nel campo dei linguaggi di programmazione. Fin dai primi passi fu chiaro che un linguaggio per applicazioni speciali doveva avere speciali caratteristiche. Già negli anni '60 iniziarono ad essere ideate e utilizzate vere e proprie generazioni di linguaggi per l'A.I., dei quali il capostipite e tuttora il più largamente usato è senza dubbio il LISP. La principale caratteristica di questo tipo di linguaggi è quella di offrire una grande flessibilità e dinamicità nello scrivere i programmi e di gestire automaticamente le risorse di basso livello, come la allocazione e deallocazione della memoria, permettendo al programmatore di concentrarsi su problemi di livello superiore. Vediamo ora quali sono le caratteristiche peculiari dei linguaggi di questo tipo.

Rappresentazione strutturale dei programmi

I programmi, nella loro rappresentazione esterna, cioè quella che costruiamo con un editor o leggiamo stampata su carta, sono testi e quindi sequenze di caratteri. Le usuali realizzazioni dei linguaggi tradizionali trattano i programmi appunto in base a questa rappresentazione. Un interprete o compilatore per linguaggi come il BASIC o il PASCAL analizza il testo dei programmi da interpretare o compilare scandendo una sequenza di caratteri. Tali compilatori e interpreti sono scritti a loro volta in linguaggi simili a BASIC e PASCAL: utilizzano quindi, per manipolare programmi, strutture dati

Una struttura di dati ad array del FORTRAN, che contiene una lista (getta la grande rossa palla) e una sottolista, fra le due parentesi. I problemi posti dall'elaborazione di liste hanno creato un linguaggio specifico, il LISP.



La stessa lista della figura precedente nella rappresentazione LISP. Ogni quadratino rappresenta un "atomo" LISP e ogni coppia di quadratini (di cui uno può essere nullo, o NIL) rappresenta un "nodo". Le frecce rappresentano i "puntatori", un altro concetto caratteristico dei linguaggi più moderni, che puntano ad un altro nodo o a una stringa di caratteri.

come i vettori di caratteri o come le stringhe alfanumeriche. In generale perciò, se ci si mantiene nell'ambito dei linguaggi tradizionali, ogni trasformazione su di un programma deve essere realizzata su questa rappresentazione. In realtà i programmi hanno una propria struttura, più sofisticata di quella rappresentabile con una sequenza di caratteri. Si pensi alla struttura implicita nei costrutti di controllo come l'IF o il FOR del BASIC (vedi figura nella pagina opposta in alto). Una istruzione che contenga un IF è rappresentabile come una struttura gerarchica il cui elemento principale identifica la struttura stessa e i cui elementi secondari sono le rappresentazioni della condizione e delle istruzioni da eseguirsi o meno dipendentemente dalla condizione stessa. Disponendo di una tale rappresentazione è molto facile individuare le azioni che l'interprete BASIC deve compiere per comportarsi come è richiesto dal programma. Riconosciuto che si tratta di una istruzione di selezione, e per questo è sufficiente consultare l'elemento principale della struttura, basta valutare la condizione e proseguire poi interpretando l'elemento opportuno della struttura. I vari elementi della struttura sono facilmente identificabili e selezionabili senza dover ricorrere al riconoscimento di parole chiave, ma semplicemente accedendo all'elemento desiderato della struttura. In realtà gli interpreti e i compilatori procedono approssimativamente in questo modo: dopo una prima analisi del programma visto come sequenza di caratteri che serve per costruire esplicitamente o implicitamente una struttura del tipo di quella vista, intraprendono la traduzione vera e propria del programma rappresentato in modo più conveniente. Un sistema di programmazione che permetta di utilizzare un linguaggio come il LISP esegue quindi, in fase di lettura, un'analisi del testo del programma simile a quella di un compilatore ma conserva del programma la rappresentazione strutturale prodotta in questo modo (come si vede nella figura qui sopra). Viene, perciò, utilizzata una struttura dati conveniente per cui sono fornite primitive di manipolazione nel linguaggio stesso. L'utente quindi può facilmente scrivere programmi agendo direttamente sulla loro rappresentazione strutturale, senza essere costretto a laboriose trasformazioni su stringhe di caratteri. È chiaro che questa caratteristica è di grande aiuto quando si desideri sviluppare sistemi che aiutano il programmatore a mantenere il controllo di grandi quantità di codice o che gli permettano di adattare la sintassi del linguaggio al problema specifico che deve affrontare. Lo scrivere programmi che aiutino a produrre e controllare altri programmi è quindi diventata una abitudine molto diffusa e amata tra i ricercatori A.I., tanto che questo genere di strumenti è diventato un vero e proprio oggetto di studio. Sono stati così sviluppati sistemi in grado di collaborare con l'utente nello sviluppo di programmi di grandi dimensioni e di alta complessità. Si tratta di ambienti che forniscono al programmatore tutti gli strumenti usuali per l'immissione del codice, la sua esecuzione e la verifica del suo corretto funzionamento ma che, oltre a questo, intervengono collaborando al disegno e all'evoluzione del programma da produrre. L'utente può ricorrere al sistema molto presto nel corso dello sviluppo comunicandogli ad alto livello di astrazione la struttura che intende dare al suo prodotto. Il sistema, in base a queste informazioni strutturali e ad altre che riesce ad ottenere esaminando il codice che via via viene aggiunto, guida interattivamente l'utente nella scelta di quali parti del programma deve produrre e nella individuazione di quali caratteristiche deve avere ogni nuova parte per essere congruente con il resto. Il sistema inoltre permette di verificare il funzionamento dei sottoprogrammi senza che sia terminato l'intero sviluppo ricostruendo, attraverso un colloquio con il programmatore, tutte le informazioni mancanti, eliminando così la noiosa necessità di scrivere codice aggiuntivo a questo scopo. Questi ambienti spesso provvedono a mantenere organizzate per l'utente tutte le informazioni sul programma che sono necessa-

rie per un corretto sviluppo. Le annotazioni su quali sono i rapporti tra le varie parti del programma, quali le scelte che hanno condizionato la stesura di un particolare algoritmo e quali sono state le più recenti e significative modifiche, richiedono nella produzione di software grandi quantità di carta e grandi sforzi di organizzazione. Se l'ambiente di programmazione "conosce" la struttura del programma, può facilmente conservare e rendere accessibili all'utente tutte queste informazioni proprio quando ne ha più bisogno, cioè mentre progetta e stende nuove parti del programma, evitandogli di perdere concentrazione trafficando con appunti conservati su carta e spesso disorganizzati.
La possibilità di manipolare i programmi in base alla loro rappresentazione strutturale rende molto facile scrivere interpreti che attribuiscano un diverso significato ai costrutti del linguaggio o definiscano costrutti nuovi. Spesso quando si sviluppano programmi che devono agire in contesti molto precisi, la definizione delle azioni che il programma dovrà eseguire viene condotta il più possibile con un linguaggio particolarmente legato al contesto. Nell'esprimere le azioni da compiere per costruire e manipolare immagini, ad esempio, è utile poter utilizzare frasi come "sposta il prisma rosso sopra alla sfera verde", senza dover dettagliare l'istruzione fino al livello del singolo punto colorato sullo schermo. Naturalmente questa esigenza è molto più impellente quando si affrontano problemi di grande complessità in contesti estremamente particolari per cui esiste da tempo un vero e proprio gergo. Tra i ricercatori A.I. si è sviluppata quindi l'abitudine, ormai diventata una tecnica vera e propria, di affrontare i problemi procedendo in due direzioni convergenti. Da un lato si cerca di analizzare il problema che si vuole risolvere e di scomporlo in sottoproblemi dettagliando poi le soluzioni parziali sempre utilizzando un linguaggio appropriato al contesto, dall'altro si sviluppa un interprete per questo linguaggio per cui i testi prodotti durante la fase di analisi siano di fatto programmi. I vantaggi di questo modo di procedere sono molti. Il lavoro è facilmente scomponibile e l'analisi del problema può essere condotta da esperti del campo specifico, che hanno quindi le idee molto chiare e non necessariamente grande competenza informatica o di A.I. La produzione dell'interprete può essere guidata da tecniche invece prettamente informatiche e per cui esiste una vasta letteratura.

Notazione funzionale

I linguaggi per l'A.I. sono per lo più basati su una notazione funzionale che sembra offrire, per i particolari tipi di programmi sviluppati in quest'ambito, concreti vantaggi. I programmi sono visti come funzioni, più o meno nel senso matematico del termine, i dati di input come argomenti e quelli di output come valori delle applicazioni delle funzioni stesse. Le azioni che i programmi devono eseguire sui dati sono quindi espresse come trasformazioni che le funzioni operano sugli argomenti per ottenere il valore.
Rimandando al futuro di questa serie di articoli qualche esempio commentato di programma scritto con una di queste notazioni, parliamo brevemente dei vantaggi di questo modo di trattare i programmi. Il vedere i sottoprogrammi come funzioni e i programmi come altre funzioni espresse come combinazione delle funzioni sottoprogrammi permette di trattare le procedure approssimativamente come strutture matematiche, utilizzando quindi terminologie e risultati ben sperimentati appunto in ambito matematico. Inoltre si possono facilmente definire funzioni che accettano come argomenti o producono come valore altre funzioni, basta ad esempio pensare agli operatori di differenziazione o di integrazione. Questo significa quindi che in un linguaggio di questo tipo si possono trattare senza particolari accorgimenti, anche dal punto di vista formale, programmi che accettano altri programmi come ingressi e producono come uscite programmi ottenuti combinandoli e manipolandoli.

Interattività

I linguaggi che normalmente si usano in ambito A.I. si presentano all'utente in modo interattivo. Il programmatore sottopone all'interprete una espressione; questa viene trasformata nella rappresentazione strutturale di cui si è parlato e quindi interpretata; il risultato di tale interpretazione viene quindi visualizzato. Tramite espressioni l'utente comunica quindi con il sistema ed ha la possibilità di specificare la definizione delle funzioni, che come si è detto sono poi i programmi, o la attivazione di funzioni già definite su dati specifici. È possibile anche associare strutture di informazioni a nomi, in modo da utilizzare il sistema come una vera e propria base di dati.

Gestione automatica delle risorse

Quando i programmi da scrivere hanno un elevato contenuto concettuale il programmatore non può permettersi di sprecare energie preoccupandosi della allocazione e deallocazione della memoria. Proprio per questa ragione i linguaggi di programmazione per l'A.I. non prevedono la dichiarazione o la richiesta esplicita della memoria che si intende utilizzare né il suo rilascio. Esiste infatti negli interpreti un meccanismo automatico che si occupa di recuperare tutte le risorse di memoria che non sono più indispensabili alla computazione o non sono utilizzate per conservare informazioni in qualche modo utili all'utente.
I nuovi linguaggi, proprio per permettere al programmatore di non preoccuparsi dei dettagli legati alla macchina che si utilizza, spesso si sono rivelati abbastanza inefficienti. La necessità di mantenere la possibilità di programmare con linguaggi ad altissimo livello ma di produrre anche programmi che realizzano grandi quantità di computazione ha condotto alla costituzione di gruppi di ricerca nel tentativo di progettare e costruire macchine la cui filosofia e architettura fosse adatta alle nuove esigenze. Prossimamente analizzeremo le caratteristiche di questo tipo di macchine e dell'ambiente che esse possono offrire all'utente.

Lezione 15

I tipi di dati

Completata questa quindicesima lezione del Corso di Programmazione e BASIC, siete in grado di eseguire gli esercizi

FRASIT.DO
FRASIP.BA

contenuti nella cassetta "5 esercizi di programmazione", lato A. I titoli seguiti dal suffisso DO corrispondono a testi, quelli seguiti da BA a programmi in BASIC.
Caricateli secondo le modalità che avete appreso.

Finora abbiamo visto come il nostro linguaggio sia in grado di elaborare informazioni di due tipi fondamentali: valori numerici e stringhe di caratteri. Abbiamo anche visto come, per indicare al calcolatore che tipo di valore sia contenuto in una certa variabile, viene differenziata la forma del suo nome, introducendo un carattere "$" alla fine degli identificatori delle stringhe.
Questo fatto si inquadra in un aspetto più ampio: dato un problema, infatti, noi ne rappresentiamo le varie entità in gioco (prezzi, velocità, nomi) con determinate variabili che possono contenere determinati tipi di valori, ma dobbiamo essere sicuri che tali tipi di valori si comportino in modo adeguato.
Facciamo un esempio. Supponiamo di voler costruire un programma che legga un numero e valuti se questo è pari o dispari; allora possiamo costruire un programma che divida per due il numero e controlli se la divisione dà resto: in caso affermativo, il numero è sicuramente dispari, altrimenti è pari.
Quindi:

```
10 ' Controllo della parita'
20 INPUT "Quale numero":N
30 ' Calcolo quoziente in Q
40 LET Q=N/2
50 ' Calcolo resto in R
60 LET R=N-Q*2
70 IF R=0 THEN PRINT "Pari" ELSE PRINT "Dispari"
80 ' Fine programma
```

Eseguendo il programma, però, abbiamo una sorpresa: tutti i numeri vengono dichiarati pari. Infatti, eseguendo con i valori 2 e 3 si ha:

```
run
Quale numero? 2
Pari
Ok
```

```
run
Quale numero? 3
Pari
Ok
```

Siamo di fronte a un programma che si comporta in modo difforme da quello che noi pensavamo ma che, dal punto di vista del calcolatore, è corretto, tanto che viene eseguito senza segnalazioni d'errore.
Ci deve essere qualche nostro errore logico, qualche "baco" (come dicono in gergo i tecnici) e, per scoprirlo, tentiamo qualche operazione di indagine della causa (di DEBUGGING).
Uno dei modi più tipici è quello di inserire nel programma delle istruzioni che stampino i risultati intermedi, in modo da capirne il comportamento.
Inseriamo, per esempio, istruzioni che stampino Q e R:

```
45 PRINT "Q=";Q
65 PRINT "R=";R
```

Eseguendo nuovamente il programma, fornendo il valore 3, osserviamo che il quoziente è dato come 1,5.
È quindi naturale che il nostro programma non funzioni: noi pensavamo di fare una divisione tra interi, e quindi di ottenere:

3 diviso 2 = 1 con resto 1

mentre il nostro calcolatore calcola anche la parte decimale del quoziente.
Da ciò impariamo che:

- diverse entità di problema devono essere rappresentate da diversi tipi di valori;
- in funzione del tipo di valore cambiano le operazioni possibili e il loro modo di agire.

Così, per esempio, se comperiamo biglietti per un teatro, saremo costretti a usare NUMERI INTERI, chiedendo 2, 3, 5 poltrone, ma non "pi greco" poltrone.
Così se vogliamo misurare la lunghezza di un tavolo, dobbiamo usare i numeri cosiddetti "reali", cioè quelli che hanno una parte decimale dopo la virgola (anzi, che hanno quella parte che può essere anche infinita, e addirittura non rappresentabile con il risultato di una frazione!) e, se troviamo una lunghezza di 2 metri, in realtà sappiamo che si tratta di 2,00 metri (non essendoci decimetri o centimetri aggiuntivi), se abbiamo usato un metro da sarto; oppure che si tratta di 2,000 metri, se riusciamo a spingerci alla misura dei millimetri, ma non sapremo se si tratta di 2,000000132 metri o di 2,00000043 metri, a meno di non usare sofisticati strumenti, sproporzionati al problema.
Ancora, per operare sui numeri interi abbiamo a disposizione le seguenti operazioni, con i relativi esempi di risultati:

+ somma	es: 3 + 2 = 5
- sottrazione	es: 3 - 2 = 1
* moltiplicazione	es: 3 * 2 = 6
/ divisione	es: 3 / 2 = 1 con resto 1
^ elevazione a potenza	es: 3 ^ 2 = 9

mentre con i numeri reali avremo:

+ somma	es: 4.5 + 3.0 = 7.5
- sottrazione	es: 4.5 - 3.0 = 1.5
* moltiplicazione	es: 4.5 * 3.0 = 13.5
/ divisione	es: 4.5 / 3.0 = 1.5
^ elevazione a potenza	es: 3.0 ^ 3.0 = 27.0

(ma si ricordi che l'elevazione a potenza non è definita se la base è negativa, mentre lo è se l'esponente è intero!).

Abbiamo cioè bisogno del concetto di TIPO di dati, che possiamo definire come:

IL TIPO DI UNA VARIABILE È UN ATTRIBUTO CHE:

- IDENTIFICA L'INSIEME DEI VALORI CHE LA VARIABILE PUO ASSUMERE
- IDENTIFICA LE OPERAZIONI CHE SU TALE VARIABILE POSSONO ESSERE EFFETTUATE
- IDENTIFICA IL COMPORTAMENTO DELLE OPERAZIONI PERMESSE

In BASIC abbiamo a disposizione ben QUATTRO TIPI DI VARIABILI:

intero
reale
reale in doppia precisione
stringhe

Abbiamo visto il senso degli INTERI: essi sono identificati dal fatto che il nome della variabile deve terminare con il carattere "%".
Se infatti riscriviamo il nostro programma precedente specificando che si tratta di variabili intere:

```
10 ' Controllo della parita'
20 INPUT "Quale numero";N%
30 ' Calcolo quoziente in Q
40 LET Q%=N%/2
50 ' Calcolo resto in R
60 LET R%=N%-Q%*2
70 IF R%=0 THEN PRINT "Pari" ELSE PRINT "Dispari"
80 ' Fine programma
```

dalla sua esecuzione otteniamo:

```
run
Quale numero? 2
Pari
Ok
```

```
run
Quale numero? 3
Dispari
Ok
```

Abbiamo già visto come le STRINGHE siano caratterizzate da un nome che termina per "$".
Abbiamo anche visto il significato delle variabili REALI; possiamo esplicitarne questa loro caratteristica facendo terminare l'identificatore con "!".
Le variabili REALI IN DOPPIA PRECISIONE si comportano esattamente come le precedenti, ma mettono a disposizione un numero di cifre significative molto maggiore, per poter avere più precisione nei calcoli: il loro nome è caratterizzato dal carattere finale "#" (questo carattere si ottiene premendo i tasti "SHIFT", "GRPH" e "H" contemporaneamente).
Tuttavia, le variabili che normalmente usiamo senza particolari specificazioni nel nome sono proprio di questo tipo.
Confrontiamo il comportamento dei vari tipi di dati per verificare le affermazioni precedenti, con il seguente programma:

Le stringhe

Le stringhe di caratteri sono legate a variabili il cui nome termina con il carattere "$".

Le loro caratteristiche sono:

lunghezza minima: 0 caratteri
lunghezza massima: 255 caratteri
occupazione: varia dinamicamente, e corrisponde a un byte (8 bit) per ogni carattere della stringa.

```
10 LET A=1/3
20 LET B%=1/3
30 LET C!=1/3
40 LET D=1/3
50 LET E$="1/3"
60 PRINT "reale       ";A
70 PRINT "intero      ";B%
80 PRINT "reale con !:";C!
90 PRINT "doppia pr.  ";D
95 PRINT "stringa     ";E$
```

L'esecuzione porta infatti a:

```
RUN
reale       : .333333333333333
intero      : 0
reale con !: .333333
doppia pr.  : .333333333333333
stringa     :1/3
Ok
```

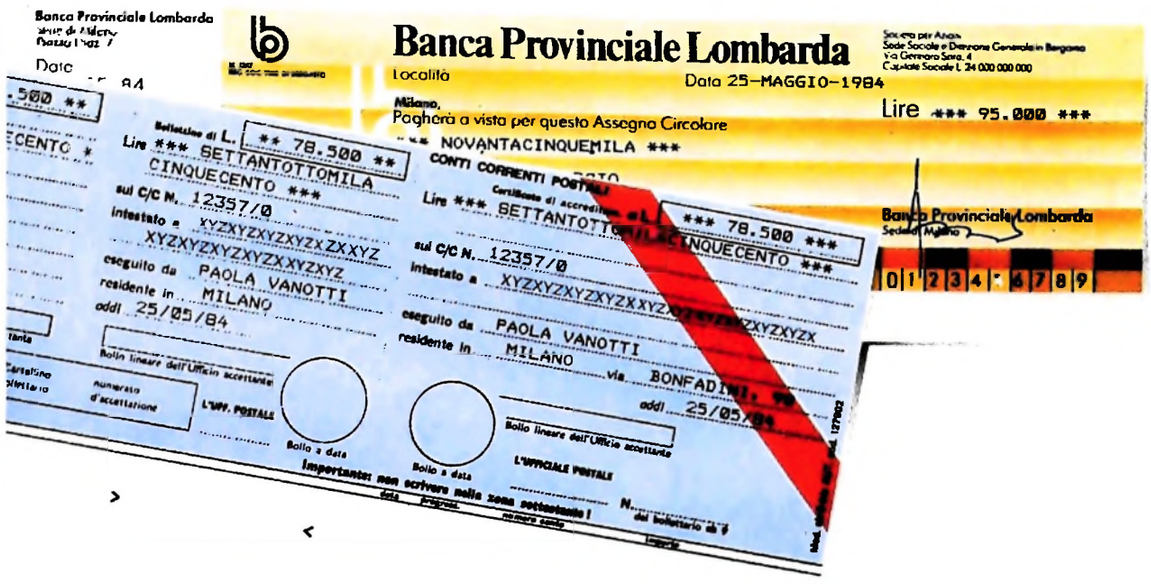
Cosa abbiamo imparato

In questa lezione abbiamo imparato:

- il concetto di tipo di dato
- il tipo intero
- il tipo stringa
- il tipo reale
- il tipo in doppia precisione
- le caratteristiche degli identificatori di ciascuno dei tipi precedenti

CONVERSIONE DA NUMERI A CARATTERI

Un programma che consente di trasformare un numero, espresso in forma decimale, nella corrispondente stringa alfabetica.



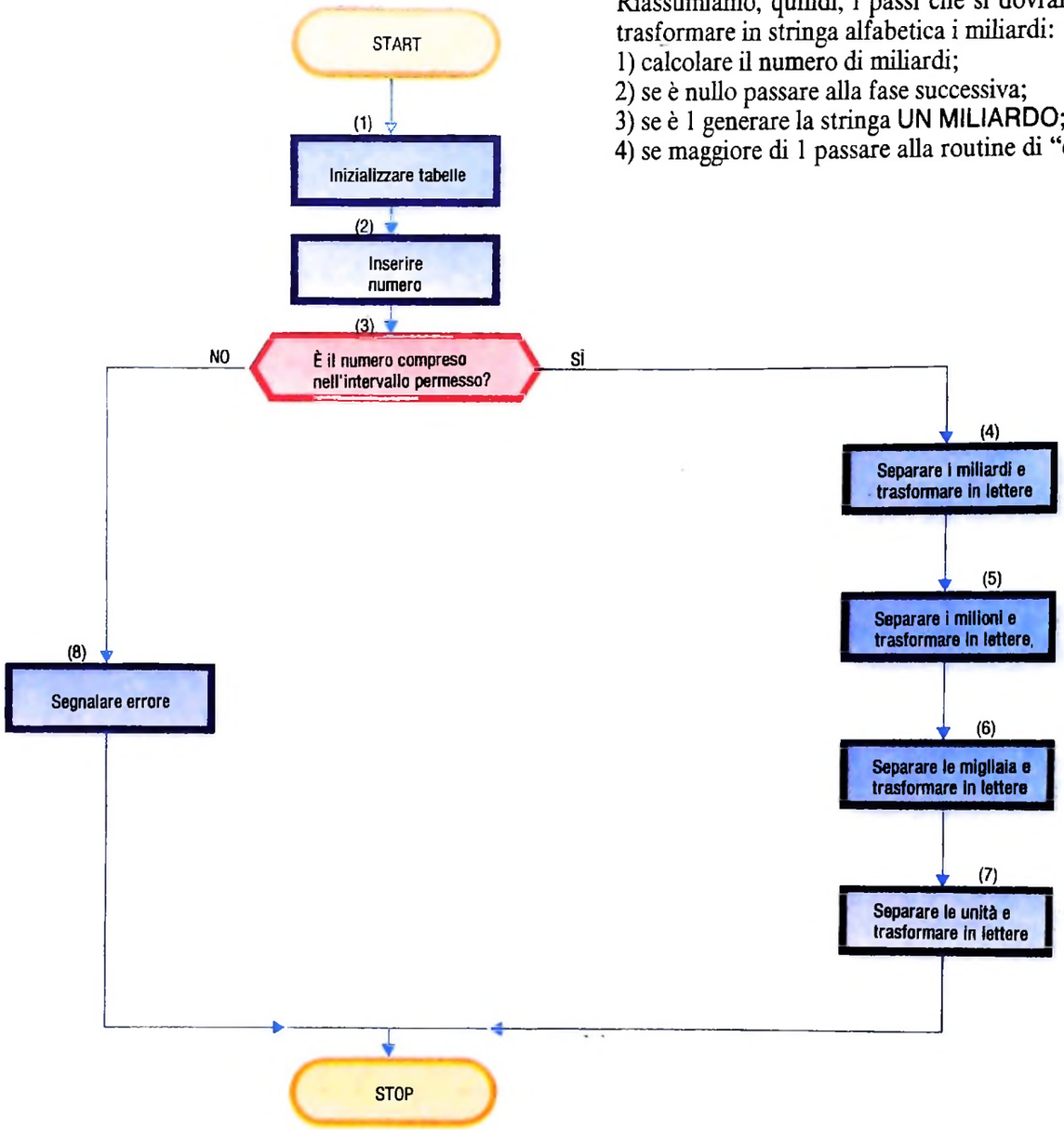
Con questo programma ci prefiggiamo di trasformare un numero, espresso in forma decimale, nella parola corrispondente formata da caratteri alfabetici, un'operazione necessaria nella compilazione di bollettini di conto corrente postale, assegni, atti notarili e in tutte le scritture nelle quali un importo deve essere scritto sia in cifre che in lettere.
Analizzeremo in dettaglio le fasi dello sviluppo del programma, dall'analisi alla stesura dei diagrammi di flusso fino alla codifica in BASIC, per rendere più chiaro e comprensibile come si affronta il problema della costruzione di un programma prefissato.

Lo sviluppo

Come prima cosa fissiamo l'intervallo di numeri entro cui il programma effettuerà la conversione: il numero da trattare dovrà essere compreso tra 1 e 999999999999. Questo non perché sia impossibile convertire numeri negativi o superiori a 1000 miliardi o comprensivi di cifre decimali, ma perché trattando di somme di denaro questi numeri di solito non compaiono.

Secondariamente, ricordiamo quali sono le regole grammaticali che regolano l'uso di aggettivi e sostantivi numerali:
a) i numeri sono rappresentati da aggettivi numerali cardinali (uno, due, ..., cento, ..., mille) e dai sostantivi miliardo, milione;
b) gli aggettivi venti, trenta, ..., novanta, perdono la vocale finale se seguiti da "uno" o "otto";
c) l'aggettivo mille ammette il plurale "mila", anche i sostantivi miliardo e milione ammettono plurale;
d) l'aggettivo uno, se seguito da un sostantivo che inizia per "m", assume la forma contratta "un".
Definite tali regole e puntualizzato il problema, dobbiamo passare all'analisi. Cominciamo con una semplice osservazione: quando, normalmente, scriviamo un numero, siamo portati a suddividerlo in gruppi di tre cifre in tre cifre, ed è quindi legittimo pensare di utilizzare lo stesso trattamento anche nel programma: prima di affrontare il problema relativo alla trasformazione di un numero espresso in cifre nella relativa forma letterale, bisognerà dunque affrontare quello di separare tra loro i miliardi, i milioni, le migliaia e le unità.
Possiamo cominciare a costruire un primo diagramma a blocchi (figura 1) che visualizzi le fasi fondamentali del lavo-

① **Prima rappresentazione del problema**



me, normale o sincopata, utilizzare per le decine.
Riassumiamo, quindi, i passi che si dovranno compiere per trasformare in stringa alfabetica i miliardi:
1) calcolare il numero di miliardi;
2) se è nullo passare alla fase successiva;
3) se è 1 generare la stringa UN MILIARDO;
4) se maggiore di 1 passare alla routine di "conversione blocco di tre cifre" (routine 9);
5) completare (a destra) la stringa alfabetica generata dalla routine 9, nel caso valga l'ipotesi precedente, con la stringa MILIARDI.

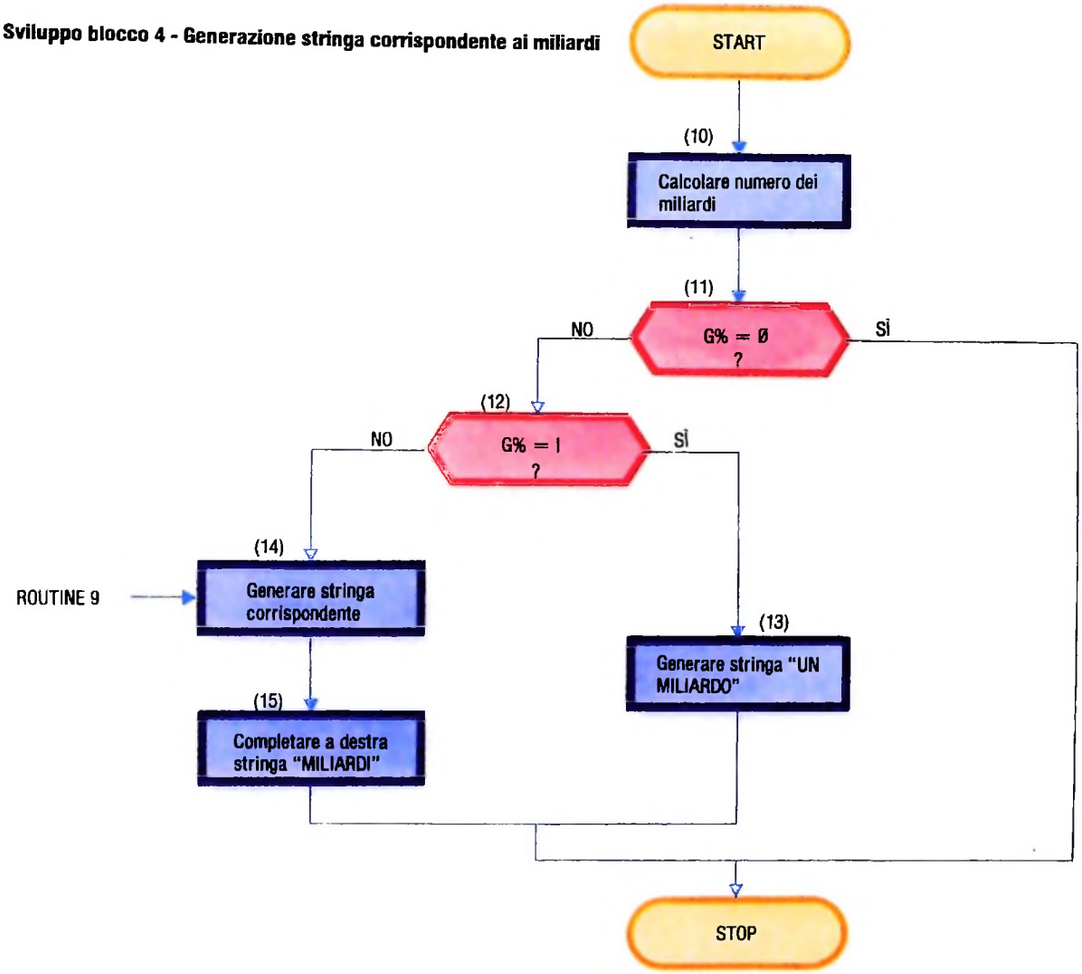
In base a tutto quanto sopra, sviluppiamo adeguatamente il blocco (4) evidenziando tutti i casi particolari enunciati (figura 2). Lo schema procedurale ottenuto è quindi composto dai seguenti blocchi logici:
10) calcolo del numero di miliardi;
11) test: il numero dei miliardi è = 0 ?;
12) test: il numero dei miliardi è = 1 ?;
13) generazione della stringa nel caso di UN MILIARDO;

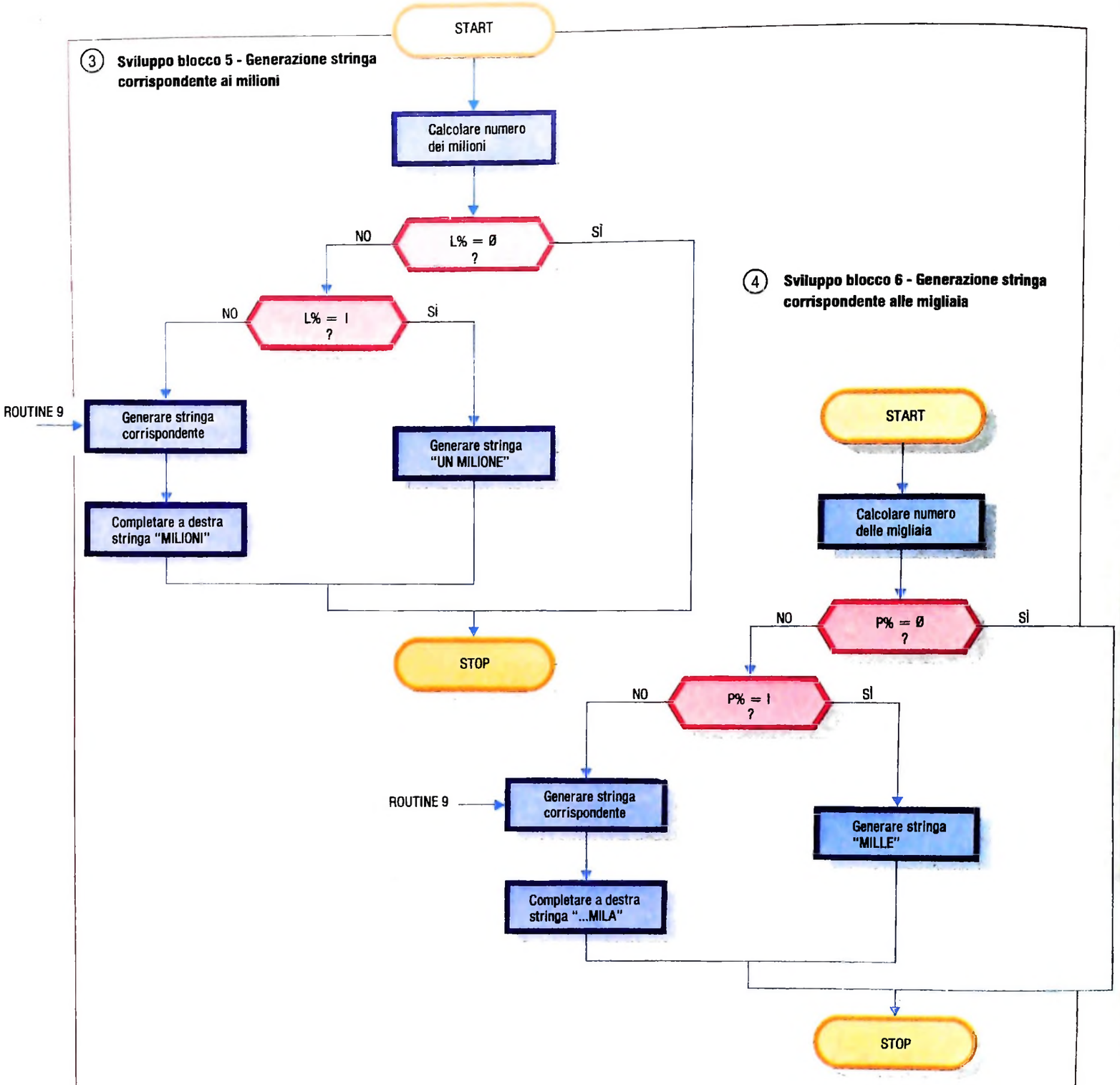
ro da svolgere. Per dettagliare maggiormente il diagramma, ripartiamo dalla definizione degli obiettivi e riesaminiamo le regole grammaticali ricordate in precedenza.
Questo ci porta a verificare che le regole a) e c) hanno carattere generale, mentre le altre vengono utilizzate solo in casi particolari, inoltre la regola b) viene utilizzata solo all'interno della routine di "conversione del gruppo di tre cifre", la regola c) viene utilizzata all'uscita di detta routine e infine la regola d) permette la generazione diretta delle stringhe "un miliardo" e "un milione". Tenendo conto di quanto dichiarato dall'ultima regola, si dovrà prevedere (nella routine succitata) un test che permetta di decidere quali tra le due for-

14) generazione della stringa, nel caso generale, con l'uso della routine 9;
15) completamento a destra della stringa risultante.
In modo analogo possiamo sviluppare i blocchi 5), 6) (figure 3 e 4).
Lo sviluppo del blocco 7) non prevede casi particolari: l'unico controllo da eseguire è quello relativo all'esistenza del valore non nullo per le unità (figura 5).
A questo punto analizziamo la già più volte citata routine 9, la "conversione del blocco di tre cifre", cioè di numeri compresi tra 1 e 999; tale routine, letto il numero, genera in risposta la stringa alfabetica che rappresenta il numero scritto in lingua italiana.
Osserviamo, per prima cosa, che nella lingua corrente esistono delle forme particolari per indicare i numeri compresi tra 1 e 19, sarà pertanto utile separare le cifre delle centinaia (variabile utilizzata C) dal blocco decine-unità (T).
Se il blocco T è inferiore a 20, sarà possibile la generazione immediata della stringa corrispondente, altrimenti occorrerà scindere ulteriormente T in cifra delle decine (V) e cifra delle unità (Z).
La routine dovrà, quindi, eseguire le seguenti operazioni per giungere alla generazione della stringa corrispondente:
- separazione della cifra centinaia dal blocco decine-unità;
- trattamento delle centinaia (C);
  * se C = 0 non genera nessuna stringa,
  * se C ≠ 0 genera la stringa CENTO preceduta (se C ≠ 1)

dalla parola corrispondente al valore di C;
- trattamento del blocco T:
  * se T = 0 nessuna generazione,
  * se T < 20 genera la stringa corrispondente al valore di T,
  * se T > 19 suddivide T in cifra delle decine V e delle unità Z e le tratta come segue:
    1) genera la stringa corrispondente al valore di V
    2) nel caso in cui Z=1 o Z=8 elide l'ultima vocale della stringa ottenuta

② **Sviluppo blocco 4 - Generazione stringa corrispondente ai miliardi**



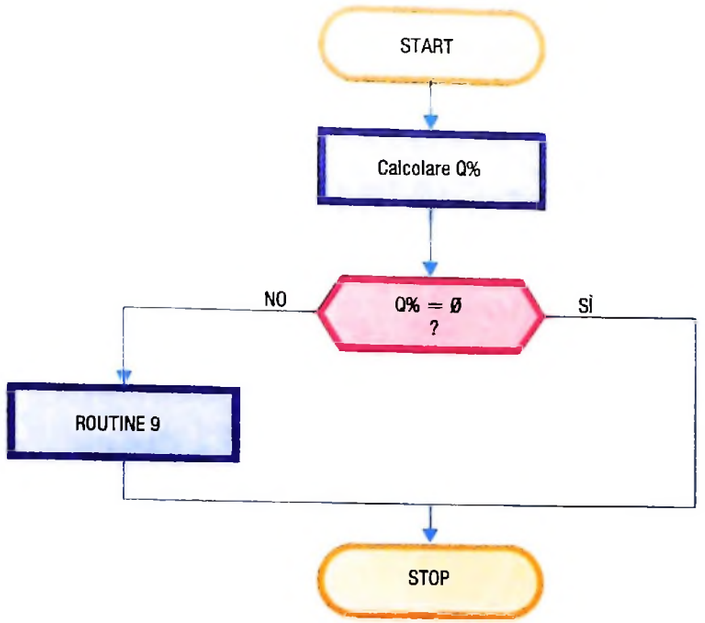


3) completa a destra aggiungendo la stringa corrispondente al valore di Z (se Z = 0 la stringa è già completa).

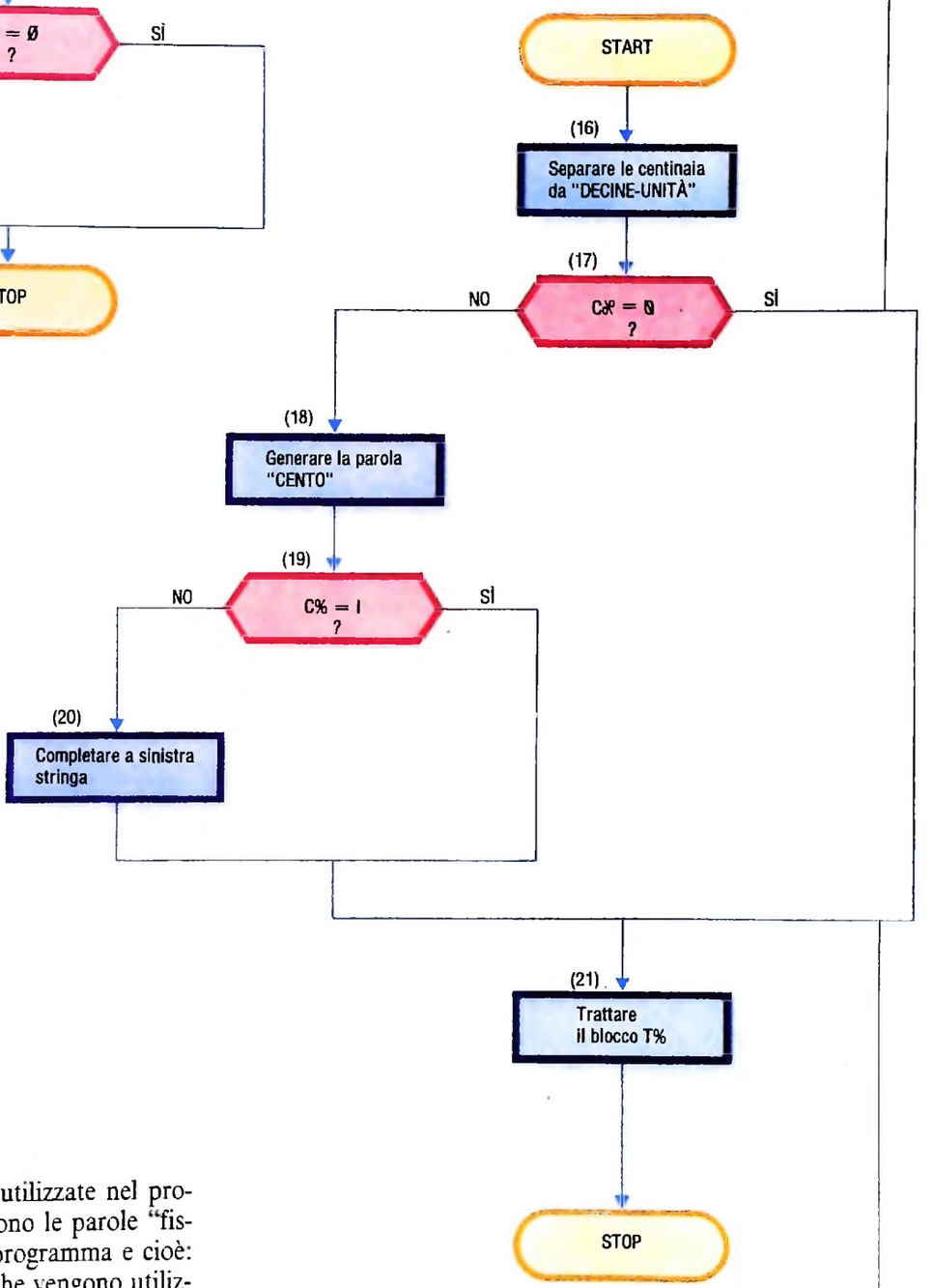
Lo schema procedurale che ne deriva è visualizzato in figura 6, mentre in figura 7 è illustrato il diagramma a blocchi relativo allo sviluppo della funzione 21), formato dai blocchi 22-32 che svolgono le seguenti funzioni:

22) test: T = 0 ?
23) test: T < 20 ?
24) genera stringa (nel caso in cui 23) sia vera)
25) scompone T in V e Z
26) genera parola troncata corrispondente al valore di V
27) test: Z = 1 o Z = 8 ?
28) test: V = 2 ?
29) completa la stringa tronca con I nel caso 28) sia vera
30) completa la stringa tronca con A negli altri casi
31) test: Z ≠ 0 ?
32) completamento della stringa con aggiunta della parola corrispondente al valore di Z.

⑤ **Sviluppo blocco 7 - Generazione stringa corrispondente alle unità**



⑥ **Sviluppo di massima della routine 9 di "conversione gruppo di tre cifre"**

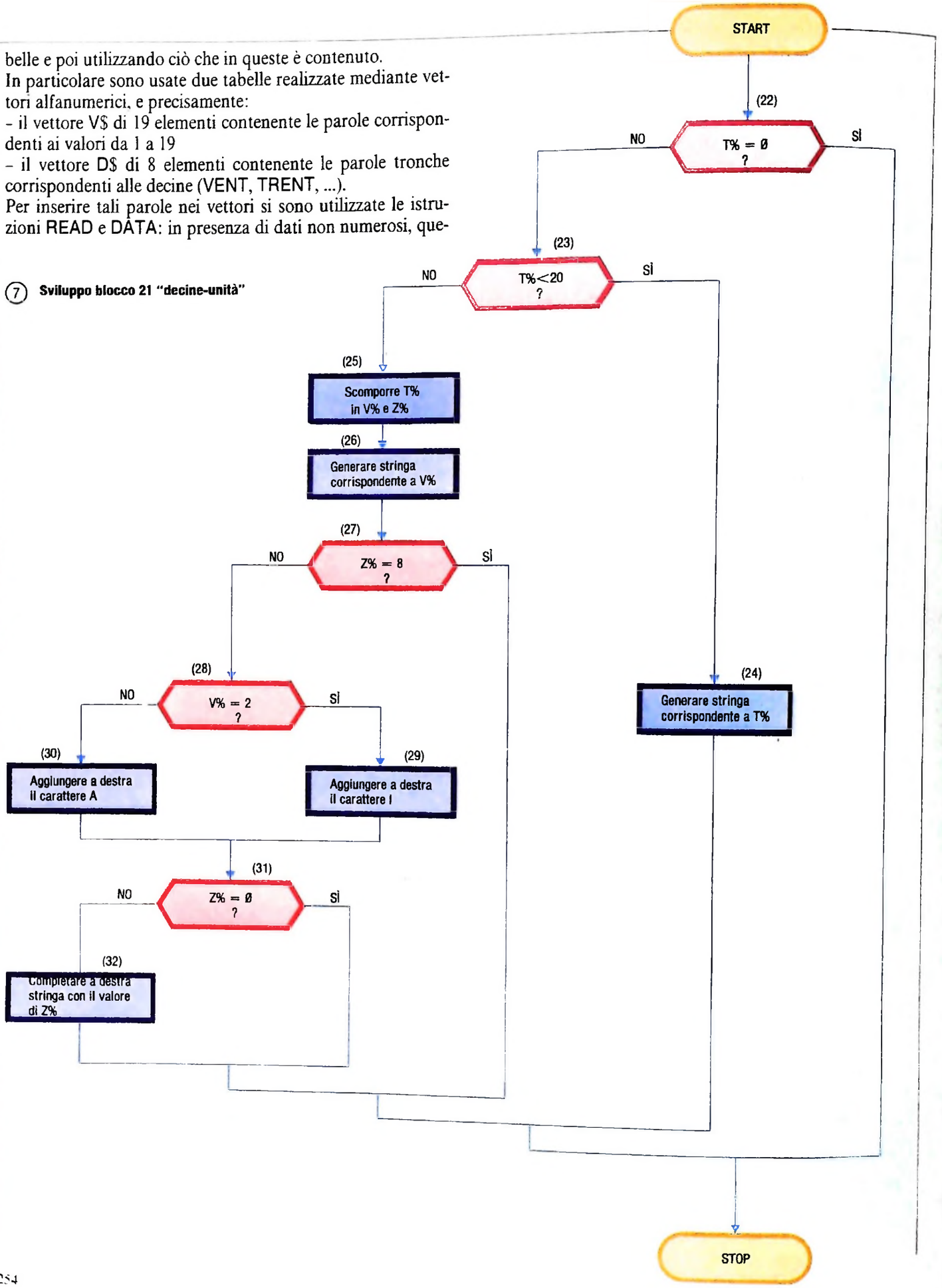


Le variabili

Esaminiamo, ora, quali sono le variabili utilizzate nel programma: innanzitutto quelle che contengono le parole "fisse" che vengono utilizzate nel corso del programma e cioè: UNO, DUE, TRE, ..., DICIASSETTE, ... che vengono utilizzate nei due diversi modi di manipolazione, e cioè *direttamente*, mediante l'uso di particolari istruzioni BASIC e *indirettamente*, memorizzando in precedenza queste parole in ta-

belle e poi utilizzando ciò che in queste è contenuto.
In particolare sono usate due tabelle realizzate mediante vettori alfanumerici, e precisamente:
- il vettore V$ di 19 elementi contenente le parole corrispondenti ai valori da 1 a 19
- il vettore D$ di 8 elementi contenente le parole tronche corrispondenti alle decine (VENT, TRENT, ...).
Per inserire tali parole nei vettori si sono utilizzate le istruzioni READ e DATA: in presenza di dati non numerosi, que-

(7) Sviluppo blocco 21 "decine-unità"



sta soluzione è migliore rispetto alla creazione di un archivio su una memoria di massa.
Occupiamoci, ora, del trattamento del numero N che viene espresso in doppia precisione per permettere l'inserimento di valori con un numero di cifre superiori a 8.
Tale valore viene associato alla variabile A ed è su questa che vengono effettuate le successive suddivisioni in numero di miliardi (G%), numero di milioni (L%), numero delle migliaia (P%), numero delle unità (Q%). Ciascuna di queste variabili potrà assumere un valore intero compreso tra 0 e 999.
Dato che di volta in volta ognuna delle variabili succitate dovrà essere trasferita alla routine 9, è utile predisporre a questo fine una variabile dello stesso tipo, chiamata variabile di trasmissione, H%. H% verrà a sua volta scomposta in cifre delle centinaia (C%) e blocco delle decine-unità (T%), che in seguito verrà suddiviso in cifra delle decine (V%) e cifra delle unità (Z%). La tabella, che vediamo qui sotto, fornisce la lista delle variabili e la loro occorrenza.

Le variabili in uso nel programma

VARIABILE	DESCRIZIONE	RIFERIMENTI
A$	stringa di stampa	210-240-280
A£	valore di N	1080-1180-1190
C$	stringa di stampa	260-280
C%	cifra delle centinaia	4030-4040-4050-4100
D$	tabella decine	100-4180
G%	numero dei miliardi	1180-1190-1200-1210-1220
H%	variabile di trasmissione	1220-1330-1440-1540-4040
I%	indice di ciclo	60-90-1030
I	variabile di lavoro	200-250
L%	numero di milioni	1290-1300-1310-1320-1330
M$	stringa risultante	170-1040-1210-1240-1320-1350-1430
		1460-4100-4150-4180-4210-4220-4230
N£	valore iniziale	120-125-1080
P%	numero delle migliaia	1400-1410-1420-1430-1440
Q%	numero delle unità	1510-1520-1530-1540
R£	variabile di lavoro	1190-1290-1300-1400-1410-1510-1520
T%	blocco decine-unità	4040-4110-4150-4160-4170
U$	tabella valori |20	50-70-4150-4230
V%	cifra delle decine	4160-4170-4180-4210
Z%	cifra delle unità	4170-4190-4200-4230
X	indice vettore	1210-1250-1320-1360-1430-1470-1560
Y$	stringa di stampa	170-190-200-210-220-240-270-280
Y	variabile di lavoro	180-190-230-240

```
10 '************************************
20 '*    CONVERSIONE CIFRE - LETTERE    *
30 '*                                   *
35 '*                                   *
39 '************************************
40 '************************************
42 '*    routine d'inizializzazione     *
45 '************************************
50 DIM U$(18)
60 FOR I%=0 TO 18
70   READ U$(I%)
80 NEXT I%
90 FOR I%=0 TO 7
100  READ D$(I%)
110 NEXT I%
115 CLS
120 INPUT"Inserire il Numero: "; N£
125 IF N£<1 OR N£>=1E+12 THEN 5000
130 GOSUB 1000
140 '************************************
150 '*  routine  stampe                 *
160 '************************************
165 PRINT " "
170 Y$=M$(0)+M$(1)+M$(2)+M$(3)
180 Y = LEN (Y$)
190 IF Y<40 THEN PRINT Y$
   : END
200 I = INSTR(Y$,"MILIONI")
```

```
210 A$ = LEFT$(Y$,I+6)
220 Y$ = RIGHT$(Y$,(Y-(I+6)))
230 Y = LEN (Y$)
240 IF Y < 40 THEN PRINT A$
   :PRINT Y$
250 I = INSTR(Y$,"MILA")
260 C$ = LEFT$(Y$,I+3)
270 Y$ = RIGHT$(Y$,(Y-(I+3)))
280 PRINT A$
   :PRINT C$
   :PRINT Y$
290 END
1000 '*********************************
1010 '*  routine di trasformazione    *
1020 '*********************************
1030 FOR I%=0 TO 3
1040   M$(I%) = ""
1050 NEXT I%
1080 A£ = N£
1150 '*********************************
1160 '*   conversione miliardi        *
1170 '*********************************
1180 G% = INT(A£/1E+09)
1190 R£ = A£ -( G% * 1E+09)
1200 IF G% = 0 THEN 1260
1210 IF G%=1 THEN M$(0)="UN MILIARDO"
    : X=X+1
    : GOTO 1260
1220 H% = G%
1230 GOSUB 4000
1240 M$(0) = M$(0)+"MILIARDI"
1250 X = X + 1
1260 '*********************************
1270 '*   conversione  milioni        *
1280 '*********************************
1290 L% = INT(R£/1E+06)
1300 R£ = R£ - L% * 1E+06
1310 IF L% = 0 THEN 1370
1320 IF L%=1 THEN M$(X)="UN MILIONE"
    : X=X+1
    : GOTO 1370
1330 H% = L%
1340 GOSUB 4000
1350 M$(X) = M$(X)+"MILIONI"
1360 X = X + 1
1370 '*********************************
1380 '*  conversione  migliaia        *
1390 '*********************************
1400 P% = INT(R£/1000)
1410 R£ = R£ - P% * 1000
1420 IF P% = 0 THEN 1480
1430 IF P%=1 THEN M$(X)="MILLE"
    : X=X+1
    : GOTO 1480
1440 H% = P%
1450 GOSUB 4000
1460 M$(X) = M$(X)+"MILA"
1470 X = X + 1
1480 '*********************************
1490 '*   conversione  unità          *
1500 '*********************************
1510 Q% = INT(R£)
1520 R£ = R£ - Q%
1530 IF Q% = 0 THEN RETURN
1540 H% = Q%
1550 GOSUB 4000
1560 X = X + 1
1570 RETURN
4000 '*********************************
4010 '*conversione gruppo di tre cifre*
4020 '*********************************
4030 C% = INT(H%/100)
4040 T% = H% - C% * 100
4050 IF C% = 0 THEN 4110
4060 '*********************************
4070 '*   analisi centinaia           *
4080 '*********************************
4090 M$(X) = "CENTO"
4100 IF C%>1 THEN M$(X) = U$(C%-1)+M$(X)
4110 IF T% = 0 THEN RETURN
4120 '*********************************
4130 '*  analisi  decine ed unità     *
4140 '*********************************
4150 IF T%<20 THEN M$(X)=M$(X)+U$(T%-1)
    :RETURN
4160 V% = INT (T%/10)
4170 Z% = T% - V% * 10
4180 M$(X) = M$(X) + D$(V%-2)
4190 IF Z% = 1 THEN 4230
4200 IF Z% = 8 THEN 4230
4210 IF V% = 2 THEN M$(X)=M$(X)+"I"
    :GOTO 4230
4220 M$(X)=M$(X)+"A"
4230 IF Z%>0 THEN M$(X)=M$(X)+U$(Z%-1)
4240 RETURN
5000 PRINT§160,"ATTENZIONE !!! Il numero
     deve essere"
5010 PRINT"compreso tra 1 e 999999999999
     e positivo"
5020 FOR I=1 TO 800
    : NEXT I
5030 GOTO 115
10000 DATA UNO,DUE,TRE,QUATTRO,CINQUE
10010 DATA SEI,SETTE,OTTO,NOVE,DIECI
10020 DATA UNDICI,DODICI,TREDICI
10030 DATA QUATTORDICI,QUINDICI
10040 DATA SEDICI,DICIASSETTE,DICIOTTO
10050 DATA DICIANNOVE,VENT,TRENT,QUARANT
10060 DATA CINQUANT,SESSANT,SETTANT
10070 DATA OTTANT,NOVANT
```

UN NUOVO MODO DI USARE LA BANCA.

Conto corrente più

TANTI PENSIERI IN MENO CON IL CONTO CORRENTE "PIU'" DEL BANCO DI ROMA.

Essere cliente del Banco di Roma vuol dire anche essere titolari del conto corrente "più". Un conto corrente più rapido: perché già nella maggior parte delle nostre filiali trovate gli operatori di sportello che vi evitano le doppie file.

Più comodo, perché potete delegare a noi tutti i vostri pagamenti ricorrenti: dai mutui all'affitto, dalle utenze alle imposte.

Più pratico, perché consente l'utilizzo del sistema di prelievo automatico Bancomat e l'ottenimento della carta di credito.

Più esclusivo, perché potete usufruire del servizio Voxintesi, attraverso il quale chiedere direttamente al nostro elaboratore il saldo del vostro conto corrente con una semplice telefonata: in qualsiasi ora come in qualsiasi giorno, anche festivo.

Più sicuro, perché con una minima spesa potrete assicurarvi contro furti e scippi mentre vi recate in banca o ne uscite.

Veniteci a trovare, ci conosceremo meglio.

BANCO DI ROMA
CONOSCIAMOCI MEGLIO.

Anche in leasing con Olivetti Leasing.

PERSONAL COMPUTER OLIVETTI M10

L'UFFICIO DA VIAGGIO

Olivetti M10 vuol dire disporre del proprio ufficio in una ventiquattrore. Perché M10 non solo produce, elabora, stampa e memorizza dati, testi e disegni, ma è anche capace di collegarsi via telefono per spedire o ricevere informazioni.

Qualunque professione sia la vostra, M10 è in grado, dovunque vi troviate, di offrirvi delle capacità di soluzione davvero molto grandi. M10: il più piccolo di una grande famiglia di personal.

olivetti

Per informazioni rivolgersi a negozi contrassegnati da "Olivetti M10 Punto di Vendita" o inviare questo coupon a: Olivetti, Direzione Personal Computer, Via Meravigli 12, 20123 Milano

NOME E COGNOME

VIA N.

CAP CITTÀ

TELEFONO