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

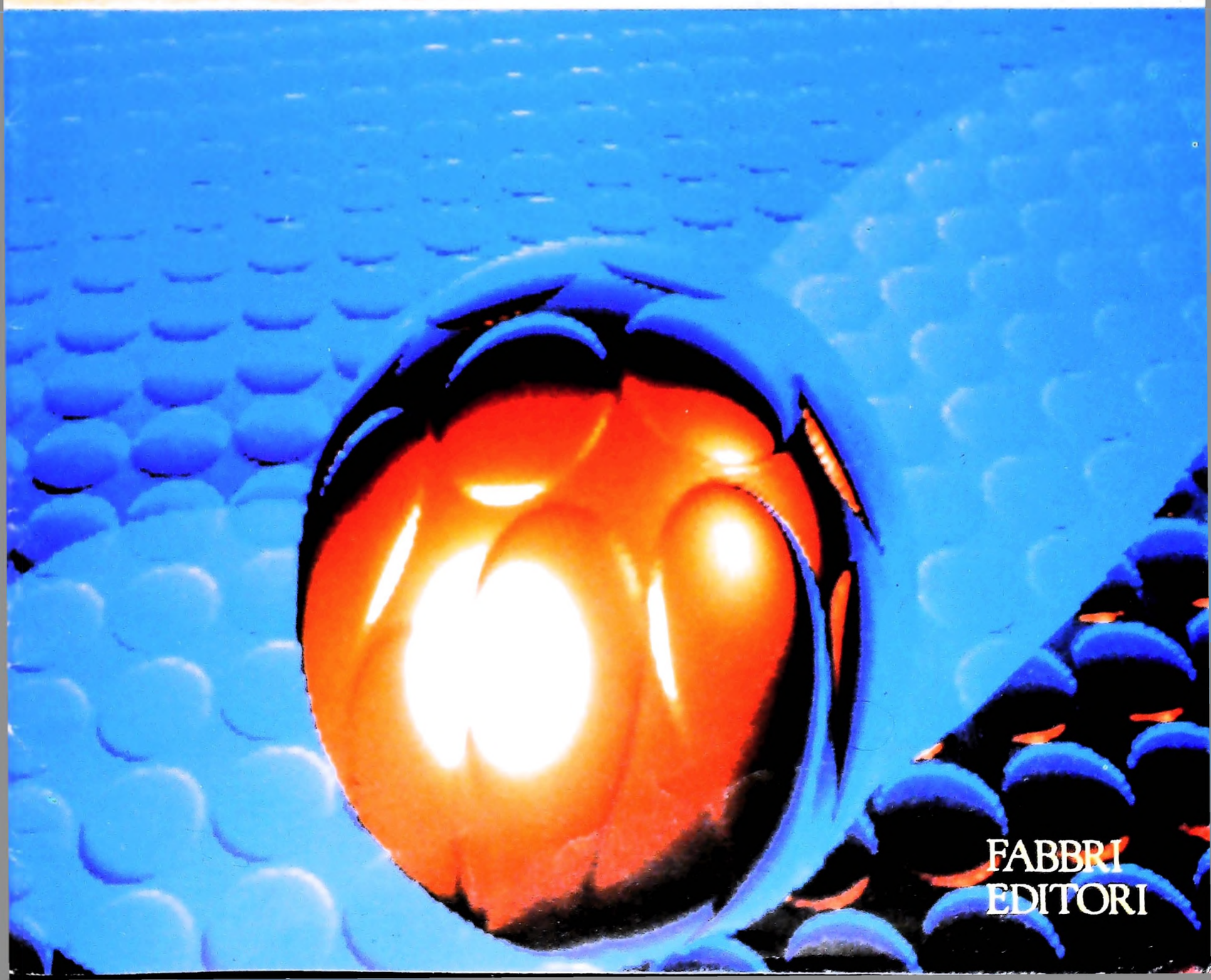
29 CORSO PRATICO COL COMPUTER

421818

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

F4 F5 F6 F7

diretto da GIANNI DEGLI ANTONI



FABBRI EDITORI

LIBRERIA DI SOFTWARE

Spediz. in abbonamento postale GR. II/70 L. 8.000 (...)

È in edicola l'undicesimo numero di LIBRERIA DI SOFTWARE dedicato a Equo canone

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
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, ENNIO PROVERA,
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

Redazione
CARLA VERGANI
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

AVVISO AI LETTORI

La prossima settimana sarà in edicola la copertina per rilegare il volume del "Corso di Programmazione e BASIC".

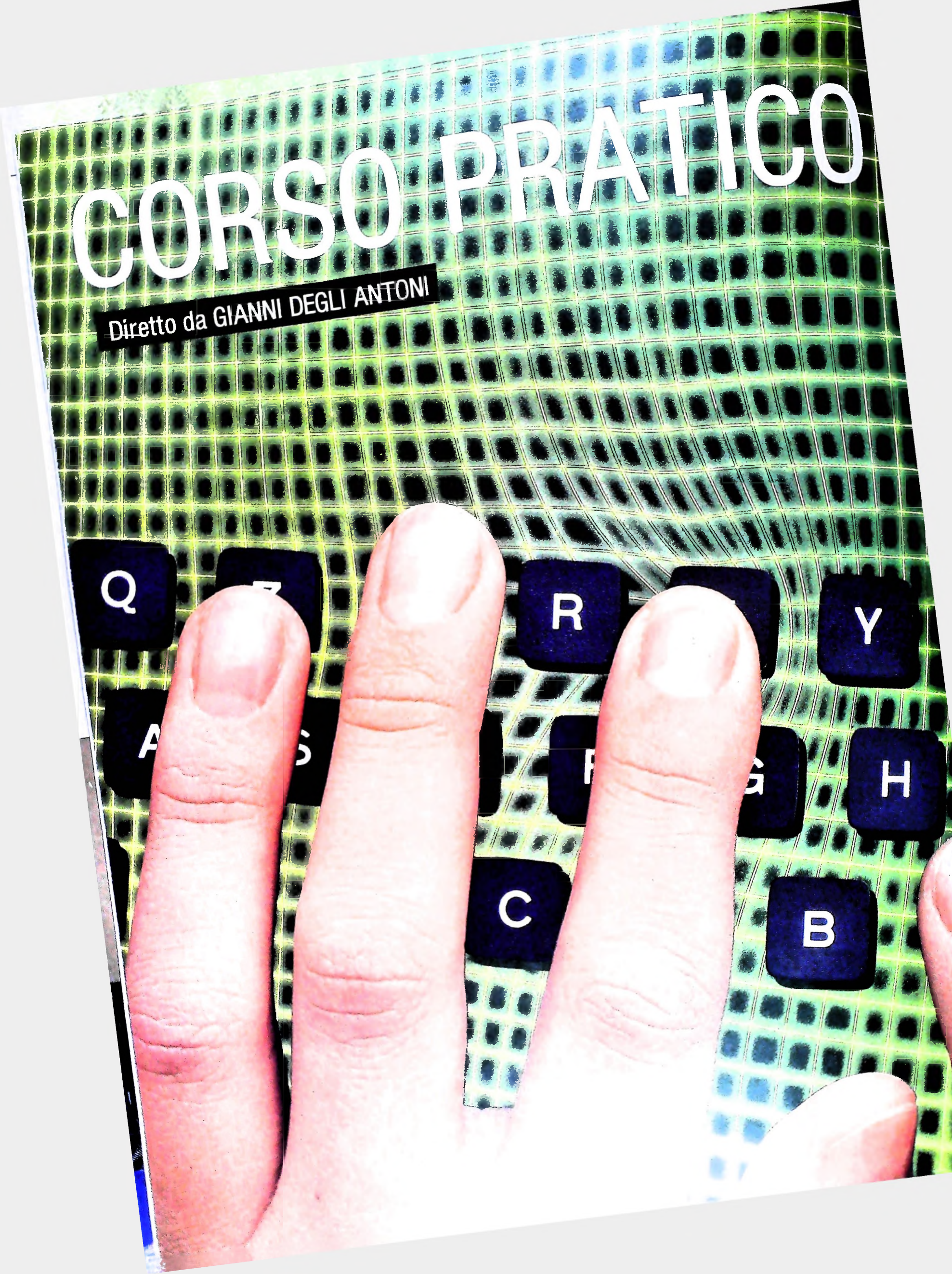
Corso Pratico col Computer - Copyright © sul fascicolo 1984 Gruppo Editoriale Fabbri, Bompiani, Sonzogno, Etas S.p.A., Milano - Copyright © sull'opera 1984 Gruppo Editoriale Fabbri, Bompiani, Sonzogno, Etas S.p.A., Milano - Prima Edizione 1984 - Direttore responsabile GIOVANNI GIOVANNINI - Registrazione 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. 29 - 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.

CORSO PRATICO COL COMPUTER

Volume Terzo

CORSO PRATICO

Diretto da GIANNI DEGLI ANTONI



COL COMPUTER

FABBRI EDITORI

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
ADRIANO DE LUCA
GOFFREDO HAUS
MARCO MAIOCCHI
DANIELE MARINI
GIANCARLO MAURI
CLAUDIO PARMELLI
ENNIO PROVERA

Direttore Editoriale
ORSOLA FENGHI

Redazione
CARLA VERGANI
LOGICAL STUDIO COMMUNICATION

Art Director
CESARE BARONI

Impaginazione
BRUNO DE CHECCHI
PAOLA ROZZA

Programmazione Editoriale
ROSANNA ZERBARINI
GIOVANNA BREGGÉ

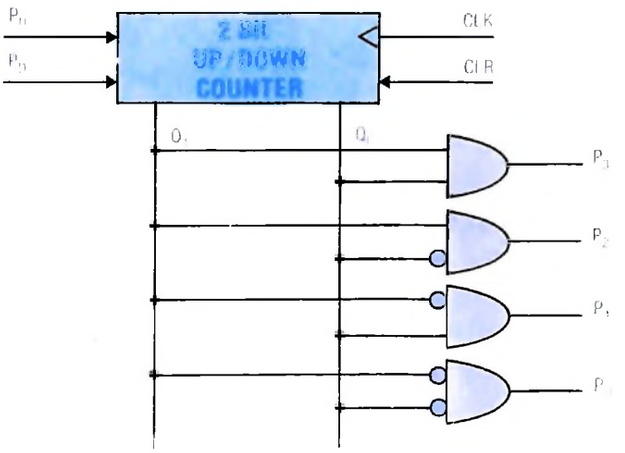
Segretarie di Redazione
RENATA FRIGOLI
LUCIA MONTANARI

Corso Pratico col Computer
Copyright © 1984 Gruppo Editoriale Fabbri, Bompiani, Sonzogno, Etas S.p.A., Milano
Prima Edizione 1984

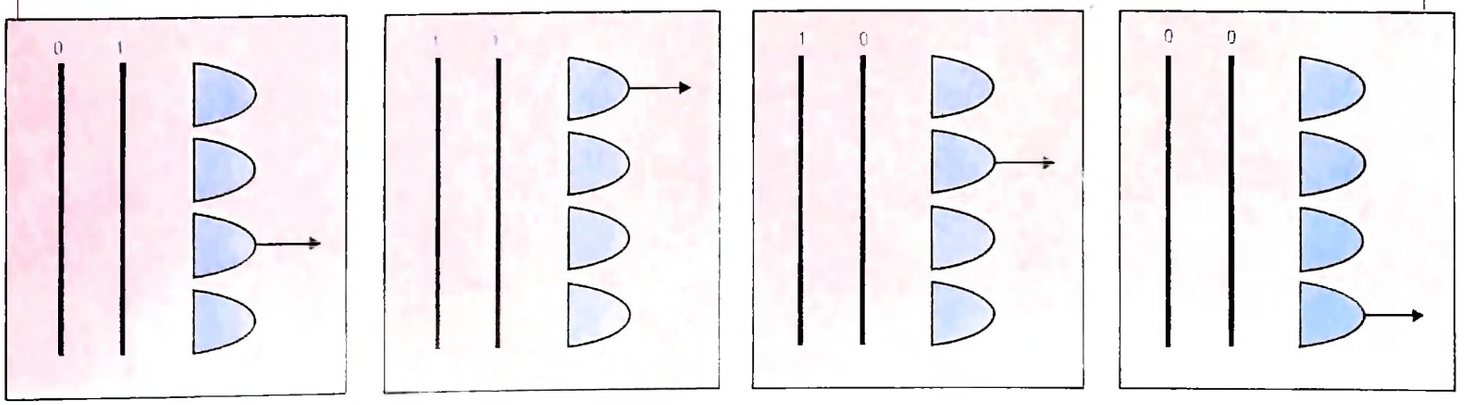
ISTRUZIONI DI SALTO "JMS XX"

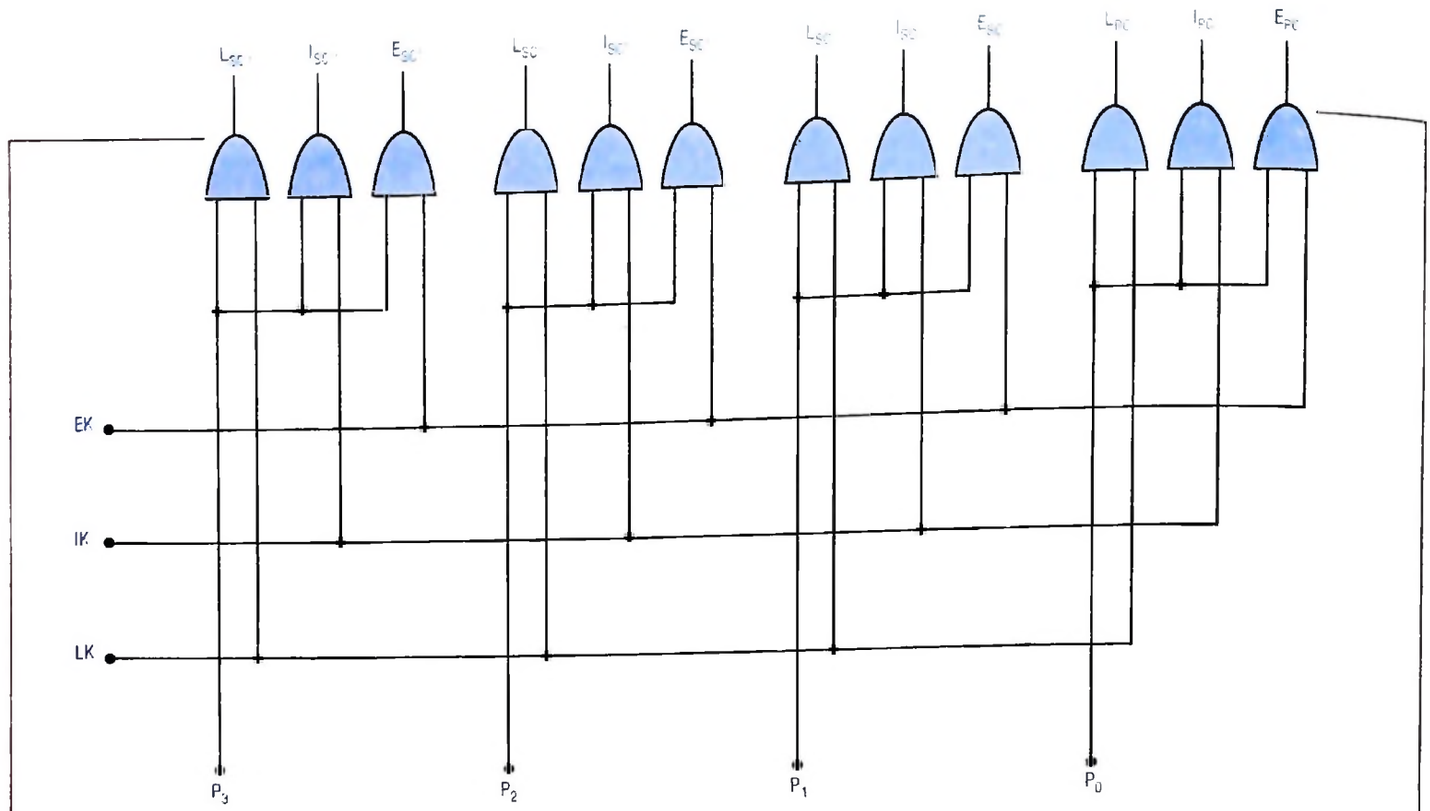
I circuiti del contatore UP/DOWN e dello STACK DEMULTIPLEXER. Dimensione della parola nell'Uamicro II.

Abbiamo visto precedentemente che ogni volta che nello svolgimento del programma si incontra l'istruzione JMS XX i comandi di esecuzione, inizialmente nel P.C., slittano al primo S.C., oppure da questo al secondo, o al terzo, mentre per l'istruzione RTS i comandi si muovono in senso inverso. Per poter realizzare questa sequenza abbiamo bisogno di un circuito chiamato "contatore UP/DOWN" capace di aumentare o diminuire il conteggio. Questo circuito, come possiamo vedere nella figura qui sotto, ha tre entrate e due uscite. L'entrata CLR, quando è attiva, azzera le due uscite Q_1 e Q_0 e attraverso la piccola matrice di AND attiva l'uscita P_0. Quando invece Pu = 1, alla transazione positiva del clock, si ha: $Q_0 = 1$ e $Q_1 = 0$, per cui P_1 è attivo; se continuiamo a tenere Pu = 1, al clock seguente, le uscite saranno $Q_0 = 0$ e $Q_1 = 1$ e P_2 sarà attivo, e, per ultimo, avremo $Q_0 = 1$ e $Q_1 = 1$ in questo caso $P_3 = 1$. Da questa sequenza possiamo dedurre che quando Pu = 1, al clock il valore binario delle uscite Q_0 e Q_1 aumenta di uno; quando invece PD = 1, al clock il valore diminuisce, per cui nella posizione $P_2 = 1$, con $Q_0 = 0$ e $Q_1 = 1$, si passa a $Q_0 = 1$ $Q_1 = 0$, scalando cioè di un posto. Dopo quest'analisi è facile capire che il comando Pu è correlato all'istruzione JMS XX, mentre il co-

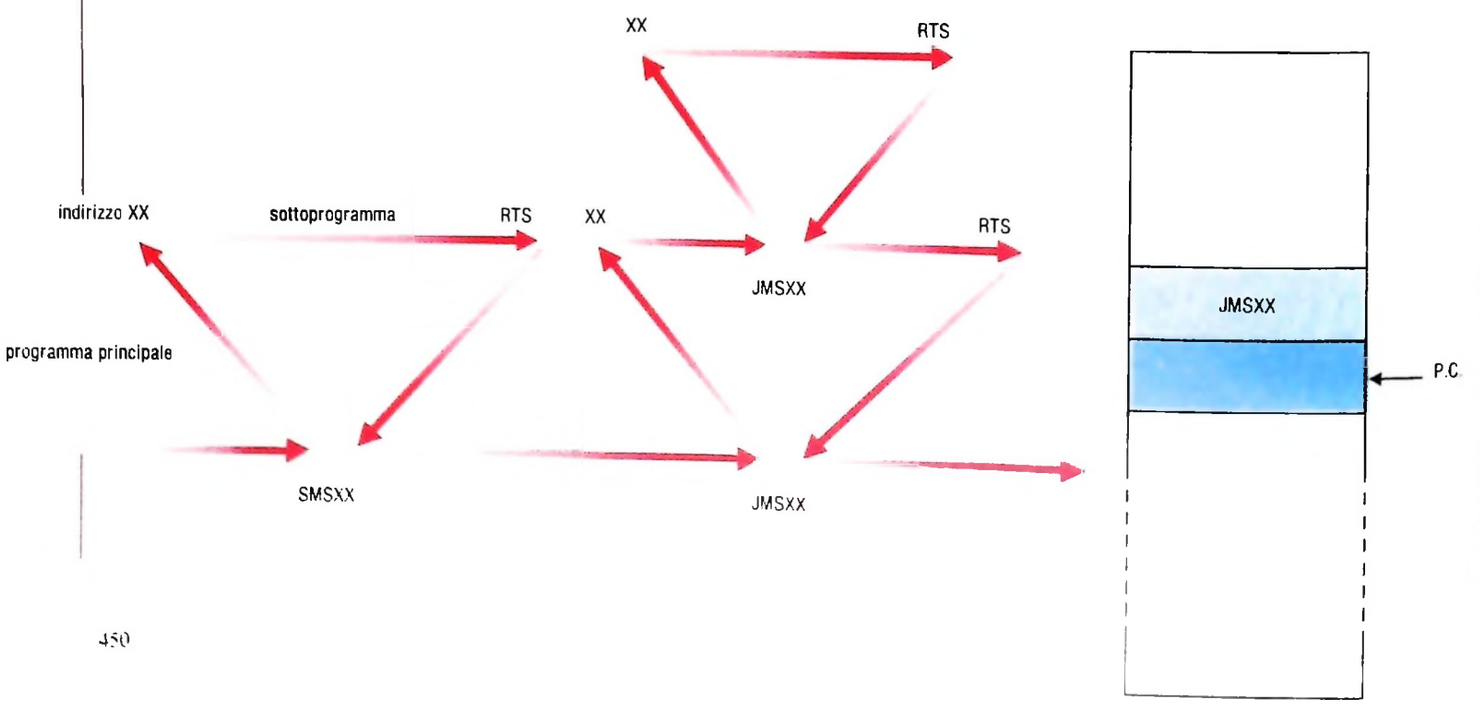


Schema di circuito del contatore UP/DOWN per il controllo dei sottoprogrammi: a lato, il circuito completo; sotto, le quattro combinazioni con le rispettive uscite attive.





mando PD all'istruzione RTS. Quando abbiamo un CLR, per esempio nel caso iniziale, il comando passa al P.C.
A questo punto manca solo un circuito per completare il quadro di tutto il registro S.C. Questo circuito è chiamato STACK DEMULTIPLEXER (figura sopra) e la sua funzione è quella di dirottare i tre segnali che vengono dal BUS di controllo: LK, IK, EK. Di questi segnali già conosciamo la funzione in quanto sono uguali a quelli degli altri registri già visti nelle lezioni precedenti, verso il registro attivo in quel momento indicato da uno dei segnali P_0, P_1, P_2 e P_3. Proviamo a ricapitolare tutto il funzionamento del circuito S.C. prendendo come base il sincronismo. Nel programma principale i comandi appartengono al registro P.C. (figura sotto a sinistra); quando incontriamo, durante la fase di FETCH naturalmente, il primo JMS XX, nel registro I.R. vengono caricati sia l'istruzione come l'indirizzo XX. L'istruzione viene decodificata immediatamente, lasciando l'operando XX inalterato, e il primo ordine emesso dal CON. è PU = 1, che incrementa il contenuto del contatore UP/DOWN per cui è al primo S.C. che vengono dirottati i comandi. Alla transizione del clock che segue viene caricato l'indirizzo XX, con LK attivo, nell'S.C. e da qui si parte per l'esecuzione del sottoprogramma, mentre nel P.C. resta l'indirizzo della istruzione che segue nel programma principale (figura sotto a destra). Quando infine incontriamo l'istruzione RTS, al termine del sottoprogramma, il segnale che viene fuori dal controllo, dopo averla decodificata, è PD = 1 e al clock i comandi passano al P.C. e da lì si riprende il programma principale; lo stesso meccanismo si ripete anche con annidamenti, come possiamo vedere nell'esempio della figura sotto a sinistra.

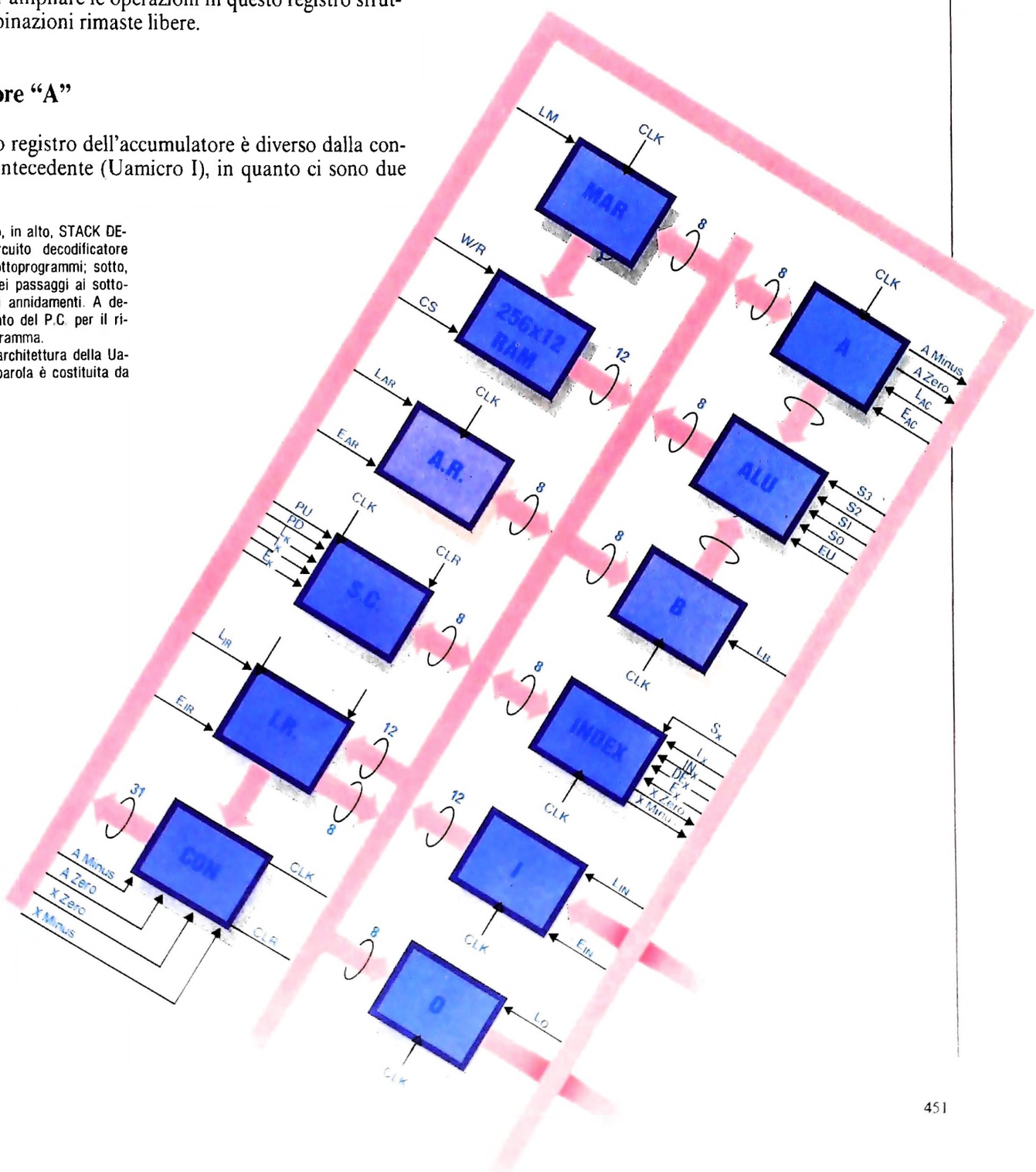


ALU

Il registro che segue per ordine d'importanza è l'ALU (figura qui sotto). Questo registro è più complesso dell'ALU della Uamicro I, perché in esso sono presenti, oltre ai circuiti per le operazioni aritmetiche, anche quelli per le operazioni logiche. Abbiamo 4 segnali di entrata: S_0, S_1, S_2 e S_3; con i quali è possibile avere 16 combinazioni per cui al limite 16 operazioni diverse; noi ci siamo limitati a creare un'architettura capace di operare su tre funzioni aritmetiche: CLR, ADD, SUB e 7 funzioni logiche $\bar{A}$, $\bar{B}$, OR, AND, NOR, NAND e XOR. Naturalmente il lettore già possiede alcuni elementi di base per poter ampliare le operazioni in questo registro sfruttando le combinazioni rimaste libere.

Accumulatore "A"

Anche questo registro dell'accumulatore è diverso dalla configurazione antecedente (Uamicro I), in quanto ci sono due segnali condizionali di uscita: AMinus quando è uguale a uno segnala che il contenuto dell'accumulatore è negativo (dei numeri negativi se ne è già parlato) e AZero quando invece è uguale a zero. Questi due segnali entrano nel controllo per completare le istruzioni di salto condizionale, che vedremo in seguito. Con il comando LAC = 1, al clock, il valore del BUS si trasferisce all'accumulatore e viene passato asincronamente nell'ALU, con il comando EAC = 1 i dati passano al BUS.



Nella pagina a lato, in alto, STACK DEMULTIPLEXER, circuito decodificatore di controllo dei sottoprogrammi; sotto, esemplificazione dei passaggi ai sottoprogrammi e degli annidamenti. A destra, posizionamento del P.C. per il ritorno da sottoprogramma.
In questa pagina architettura della Uamicro II, dove la parola è costituita da 12 bit.

La parola e la sua dimensione

Prima di proseguire l'esame dei vari registri dell'Uamicro II, facciamo una breve disgressione su un aspetto rilevante della Uamicro II. Avevamo visto nella versione precedente (Uamicro I), come la parola avesse la stessa quantità di bit (8 bit) sia per le istruzioni sia per gli indirizzi e dati, invece adesso le cose sono cambiate sia in grandezza sia in organizzazione. La parola è di 12 bit e può essere interpretata in diversi modi: divisa in due parti, una per l'istruzione l'altra per l'operando, oppure una parte indica il tipo d'istruzione (per esempio le istruzioni di operazioni) e l'altra l'istruzione stessa; per ultimo la stessa parola può indicare un semplice dato. Tutta questa flessibilità di interpretazione della parola in memoria, anche se aumenta considerevolmente la complessità del sistema e della sua architettura in compenso ne aumenta l'efficienza e la velocità, caratteristiche queste sempre più necessarie nei sistemi moderni.

Accumulatore "B"

È un registro di 8 bit tale che, con il comando LB =1, il valore del BUS passa al registro e da qui all'ALU come per l'accumulatore "A". Non è presente il comando EB per cui i dati non possono passare al BUS.

MAR

È un registro sincrono che immagazzina 8 bit dell'indirizzo di memoria. La sua funzione è simile al MAR della versione precedente (Uamicro I).

MEMORIA

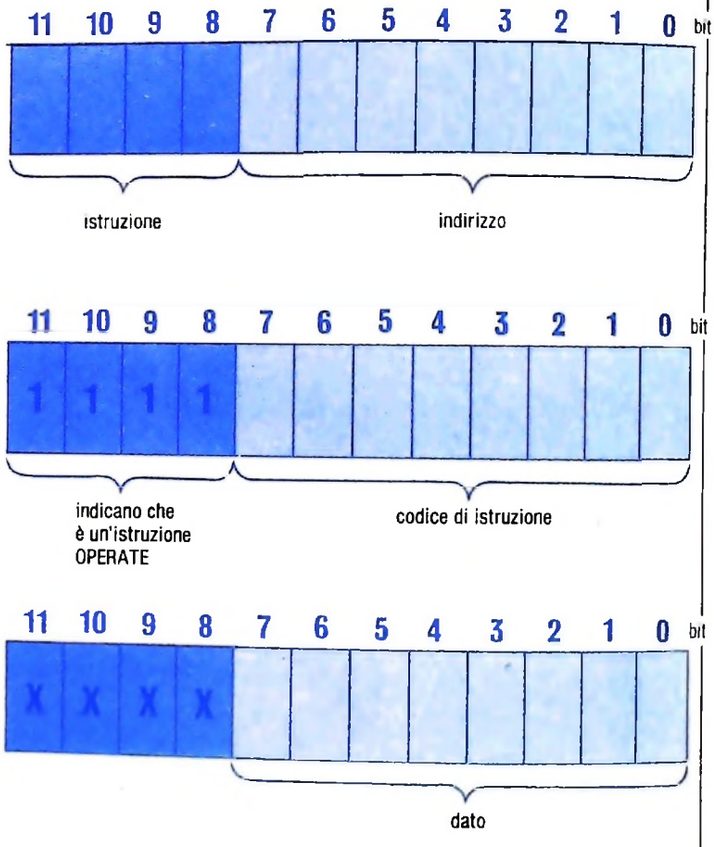
La memoria è composta da 256 parole di 12 bit, i cui indirizzi vanno da OOH a FFH. Il segnale CS (selezione del circuito) viene direttamente dal controllo. Quest'ultimo segnale è comune a quasi tutti i microprocessori, naturalmente con nomi diversi, per esempio $\overline{\text{MREQ}}$ o $\overline{\text{VMA}}$ come nella versione precedente. Da notare che in questa configurazione, essendo la memoria separata dai registri di entrata e uscita, non è stata necessaria la creazione del circuito decodificatore perché gli stessi comandi provenienti dal controllo fanno la separazione dei registri. Con il comando $W/\bar{R} = 1$ e $CS = 1$ si effettua una lettura mentre con $W/\bar{R} = 0$ e $CS = 1$ si effettua una scrittura. Da ricordare che anche in questo caso la scrittura è una operazione sincrona con lo stesso segnale $W/\bar{R}$.

A.R. (Registro ausiliario)

È un registro ausiliario, di proposito generale, molto utile nello sviluppo delle operazioni di esecuzione.

I.R. (Registro di istruzioni)

Con la Uamicro II il formato della parola è di 12 bit e la parola può assumere diversi significati secondo la configurazione. Per le istruzioni con riferimento alla memoria, tipo MRI (figura in alto), nei quattro bit più significativi sono immagazzinati i codici delle istruzioni mentre negli 8 bit meno significativi gli indirizzi di memoria. Esiste una convenzione, ormai generalizzata, che è quella d'indicare il bit più significativo, che occupa cioè la posizione media, con la sigla MSB e il meno significativo, che occupa la posizione zero, con la sigla LSB. Con le istruzioni di operazioni "OPERATE" (figura al centro) i 4 bit più alti sono tutti 1 per indicare che gli 8 bit più bassi rappresentano il codice di un'operazione per i registri interni. Per completare il quadro, quando il contenuto della memoria è un dato i bit validi sono gli 8 bit più bassi, mentre i 4 più alti vengono tralasciati (figura in basso).



CON. (Controllo)

Questo registro ha la stessa funzione della versione precedente (Uamicro I); la differenza consiste nella maggiore quantità di linee di controllo che aumentano a 38, come era prevedibile, e per la presenza di 4 linee condizionali provenienti da due registri interni. Una considerazione: le linee di controllo interno occupano una parte considerevole dello spazio del circuito e aumentano proporzionalmente all'aumentare del numero dei registri interni o della loro complessità.

Il concetto di file

Uno degli impieghi principali degli elaboratori è il trattamento di grandi quantità di dati, che presentano in generale la stessa struttura e richiedono lo stesso tipo di elaborazione. Si pensi per esempio a problemi tipici come la compilazione e la stampa delle fatture o la gestione delle paghe e degli stipendi, dove le entità in gioco sono sempre le stesse e così pure il tipo di elaborazione, ma quello che varia sono i dati specifici di ogni fattura o di ogni stipendio.
Per trattare tali tipi di problemi occorre disporre di opportune strutture di dati che consentano sostanzialmente di costruire veri e propri "elenchi" dei dati da trattare. Così nel caso delle fatture bisognerà avere a disposizione l'elenco di tutti i clienti e dei servizi forniti a ciascuno di essi. Si tratta di avere vere e proprie "file" di dati a cui i programmi accedono per attingere di volta in volta le informazioni da trattare.
Una struttura di questo tipo è già stata vista ed è la struttura array, che però, nella versione Pascal presentata, prevede un numero definito a priori di elementi. Nel caso invece di problemi quali i suddetti, il numero di informazioni da trattare non è noto a priori. Inoltre molto spesso la quantità di dati da elaborare è tale da non trovare spazio sufficiente in memoria. Per questo storicamente si è posto il problema di poter usare supporti di memoria ausiliari oltre alla memoria primaria dell'elaboratore. Da questa esigenza sono pertanto nati i vari tipi di memorie secondarie (dischi, nastri, floppy...) e le relative periferiche (unità a disco, a nastro, driver per floppy). Da tali problemi è nato il concetto di "file" ovvero di "fila" di dati, che tradizionalmente risiede su una memoria secondaria.

Il costruttore di tipo "file"

Il differente supporto usato per i file rispetto alle altre strutture dati residenti in memoria principale rende molto più complesse le modalità di accesso ai dati in essi contenuti. In questo caso infatti si tratta di creare un canale di comunicazione tra il calcolatore e la periferica relativa, che è spesso a sua volta un calcolatore specializzato nel trattamento di uno specifico tipo di supporto. Per questo motivo i file vengono trattati come strutture di dati a sé.
Il Pascal grazie al concetto di costruttore di tipo ha abolito questa diversificazione. Esiste pertanto il costruttore FILE che è definito come un insieme di elementi omogenei (cioè dello stesso tipo) il cui numero non è noto.
Come per i costruttori già visti, la dichiarazione di una variabile file deve specificare qual è il tipo dei componenti, che possono essere sia di tipo semplice sia di tipo complesso. Così è possibile dichiarare variabili come le seguenti:

```
VAR INTERI: FILE OF INTEGER;
    TESTO: FILE OF CHAR;
    COLORI: FILE OF (BLU, ROSSO, GIALLO, VERDE, NERO, BIANCO);
```

che dichiarano al compilatore rispettivamente l'uso di un file di interi, di un file di caratteri e di un file di colori.
Sono però ammesse anche dichiarazioni come le seguenti:

```
VAR ANAGRAFICHE: FILE OF RECORD
      NOME: STRINGA;
      ETÁ: 0..100;
      SESSO: (F,M);
      INDIRIZZO: STRINGA
                  END;
```

che dichiara l'uso di un file i cui elementi sono record e la seguente:

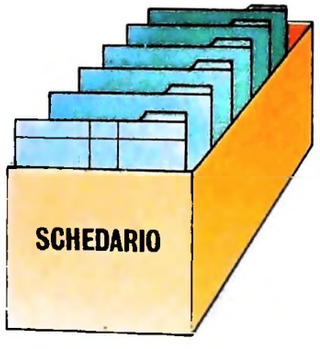
Come si vede dalle precedenti dichiarative è possibile costruire strutture complesse a partire dal costruttore FILE in modo analogo a quello visto per i costruttori array e record.

```
VAR TEMPERATURE: FILE OF ARRAY [1..7] OF-40..50;
```

che dichiara un file i cui elementi sono array relativi alle temperature giornaliere per tutti i giorni della settimana.
Le strutture così descritte prescindono dal supporto di memoria su cui verranno allocate. La visione del linguaggio Pascal è cioè ancora una volta rivolta agli aspetti logici piuttosto che a quelli più strettamente fisici e legati ai problemi tecnici di costruzione dei programmi (si dice in gergo "implementativi"). In sostanza dal punto di vista della definizione del problema e di un modello di struttura dati a esso omogeneo, non hanno interesse i vincoli "fisici" di implementazione, che saranno problemi da risolvere in un momento successivo quando dalla struttura logica dei programmi si passerà ai programmi veri e propri scritti nel linguaggio disponibile sul nostro elaboratore.
In effetti le strutture così descritte possono virtualmente risiedere sia in memoria principale sia in memoria secondaria. Vedremo come alcuni linguaggi offrano effettivamente entrambe queste possibilità.

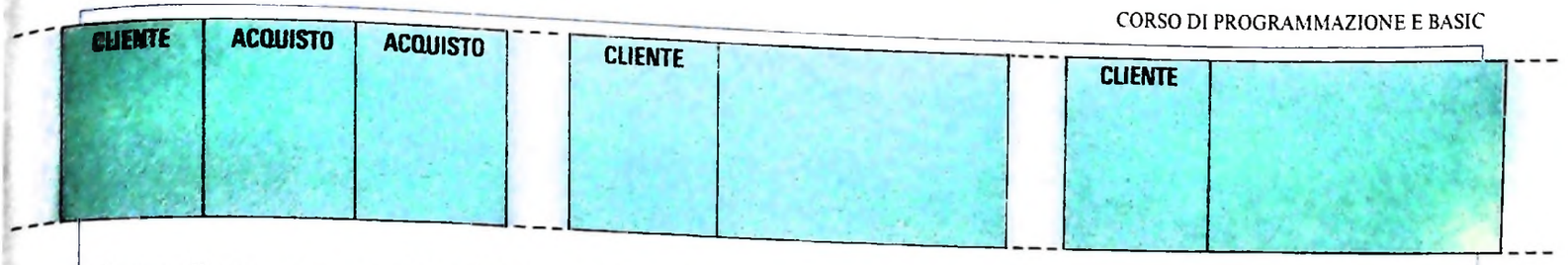
Struttura file: un esempio

Per meglio chiarire il concetto di file anche da un punto di vista intuitivo, proviamo a immaginare lo schedario di un fornitore nel quale ogni scheda contiene il nome di un cliente e le indicazioni degli acquisti da questo effettuati e del valore di ciascuno di essi e inoltre il valore totale corrispondente:



CLIENTE	Cod	PREZZO
Rossi	03 05	10.000 30.000
		40.000

Se adesso riportiamo lo stesso schedario su un supporto magnetico, per esempio un nastro, potremo rappresentarlo così:



dove vediamo una successione di "elementi" corrispondenti ciascuno a una scheda.
Se adesso descriviamo la struttura mostrata lo faremo così:

```
VAR SCHEDARIO: FILE OF
        RECORD
            CLIENTE: STRINGA;
            ACQUISTI: ARRAY 1..N OF
                RECORD
                    CODART: 1..1000;
                    IMPORTO: 1000..1000000
                END;
END.
```

Proviamo allora a costruire lo schema del programma che calcola il totale dei ricavi di tale fornitore. Come al solito usiamo un linguaggio che si ispira al Pascal, anche se non ne rispetta tutte le regole:

```
PROGRAM TOTALEGENERALE;
    BEGIN
        TOTALE: = 0;
        leggischeda;
        WHILE scheda presente DO
            BEGIN
                TOTALE: = TOTALE + SCHEDARIO↑. TOTALE;
                leggischeda
            END;
        stampatotale
    END.
```

Si noti che nel programma vengono usate in realtà due variabili di nome TOTALE; non si crea però nessuna ambiguità poiché l'una è una variabile semplice (quella utilizzata per totalizzare il risultato finale), mentre l'altra è un campo di un record. In particolare è un campo del record letto dal file e, trattandosi di una struttura record, l'accesso avviene secondo le regole viste e cioè indicando il nome del record che è uguale a quello del file) seguito da "." e dal nome del campo. Inoltre compare il simbolo "↑" a indicare che si tratta del record del file di identico nome.

Si noti anche che per maggior chiarezza è stata usata la sintassi di dichiarazione di un programma Pascal nell'intestazione (la parola chiave PROGRAM seguita dal nome del programma e la coppia BEGIN END per isolare gruppi di istruzioni interne a una stessa struttura. Il "." dopo l'ultima END indica la fine del testo del programma).

Modalità di accesso ai dati in file

Come per gli altri costruttori, dobbiamo definire anche per il FILE le modalità di accesso ai componenti. I componenti di una struttura file in Pascal sono generalmente accessibili solo in sequenza. Ciò significa che l'accesso avviene sempre partendo dall'inizio e scandendo successivamente elemento per elemento. Non è possibile quindi fare riferimento diretto a un componente anche qualora si conoscesse la sua posizione. Pertanto ogniqualvolta si voglia trattare un elemento precedente a quello su cui siamo attualmente posizionati, ci troveremo costretti a riposizionarci all'inizio del file e a scandirlo nuovamente fino a raggiungere l'elemento ricercato.
Questo vincolo alle modalità di accesso potrà sembrare un limite, ma ha una sua giustificazione se pensiamo che file, nel senso intuitivo di "file" di dati, sono anche la stampante e la tastiera dove l'ordine dei dati è evidentemente sequenziale. Né possiamo immaginare di "muoverci" sui dati stampati secondo un ordine "casuale", che implicherebbe un movimento in avanti e indietro del rullo della stampante stessa; né, analogamente, possiamo pensare a un ordine diverso nel caso dell'introdu-

zione di dati da tastiere, dove l'ordine sequenziale è vincolato dalla sequenza temporale con cui digitiamo i caratteri. Se però torniamo all'esempio dello schedario e ripensiamo all'immagine della sua collocazione su un nastro, non ci sarà difficile capire come il modo più naturale per "raggiungere" un'informazione sia effettivamente una scansione sequenziale del nastro stesso, che corrisponde al modo normale di trattare anche i nastri che ci sono più abituali e cioè quelli dei registratori.

La necessità di ricorrere a primitive d'accesso è determinata dal fatto che i file risiedono generalmente su una memoria secondaria. Ciò comporta che ogni operazione di lettura o scrittura implica in realtà un'operazione di trasferimento dal calcolatore alla periferica interessata o viceversa con invio di comandi a organi meccanici. Questa operazione è molto complessa e richiede una accurata conoscenza sia del calcolatore che della periferica anche da un punto di vista di struttura fisica. Per questo tutti i linguaggi di programmazione di alto livello mettono a disposizione "istruzioni" ad hoc che il compilatore provvede poi a espandere nell'insieme di istruzioni di basso livello necessarie per effettuare il trasferimento.

L'accesso al singolo componente è effettuato tramite "primitive" che, nel caso del Pascal sono GET e PUT rispettivamente in lettura e scrittura. Si tratta di "procedure" ovvero sottoprogrammi già disponibili nel linguaggio e direttamente richiamabili senza bisogno di ulteriori specifiche. Pertanto se vogliamo leggere il valore del componente attuale di un file F diremo: GET(F)
e se vogliamo "aggiungere" un nuovo componente diremo: PUT(F)
Poiché le operazioni di lettura e scrittura da e sul file implicano un trasferimento fisico di dati è necessario disporre di un'area d'appoggio in cui:

- viene deposto il valore che si vuole scrivere sul file;
- viene deposto il valore letto dal file.

Si tratta di una variabile a tutti gli effetti, che viene denominata "buffer". In Pascal l'uso di una variabile di tipo FILE implica automaticamente l'esistenza di una variabile buffer del tipo dei componenti del file. Tale variabile non necessita di una dichiarazione esplicita, ma viene automaticamente generata dal compilatore quando incontra una dichiarativa di tipo file. Se il file dichiarato si chiama F il buffer generato viene indicato come F^.
Un'operazione di aggiunta di un nuovo elemento al file F deve quindi essere preceduta da un'operazione di assegnamento alla variabile F^. Se quindi F è per esempio un file di interi, un'operazione di scrittura dovrà essere preceduta da un'istruzione di assegnamento del tipo: F^: = 10; F^: = V;
dove V è una variabile numerica.
La successiva: PUT(F)
avrà così l'effetto di accodare il valore assegnato a F^ in fondo al file.
In modo analogo l'istruzione: GET(F)
ha l'effetto di rendere disponibile nella variabile F^ il valore dell'elemento del file F su cui si era posizionati.
Si noti che ogni accesso tramite le suddette primitive causa un avanzamento automatico sull'elemento successivo. In tal modo l'operazione seguente sarà automaticamente effettuata sul componente successivo, senza operazioni aggiuntive.

Cosa abbiamo imparato

In questa lezione abbiamo visto:

- Il concetto di file come "fila" di informazioni normalmente residente su una memoria secondaria;
- il costruttore di tipo "FILE" in Pascal;
- le modalità di accesso ai componenti di una struttura file in Pascal (GET e PUT);
- il concetto di "buffer".

L'INFORMATICA E LA GESTIONE DELLA CLASSE

La gestione "informatica" della classe permette all'insegnante di svolgere il suo ruolo in modo totalmente nuovo, rivoluzionario rispetto alla visione tradizionale.

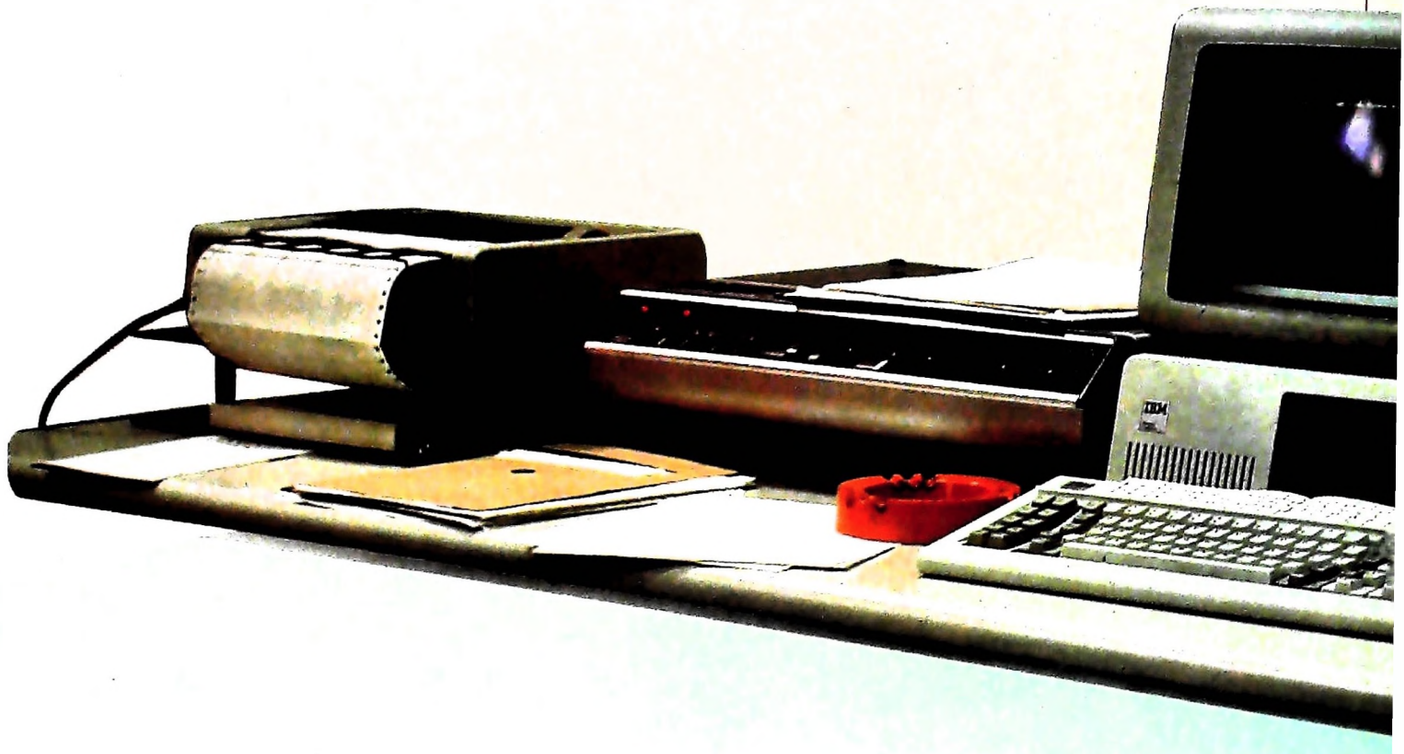
Come abbiamo già visto, l'applicazione alla didattica di strumenti e metodi dell'informatica è piuttosto vasta e tuttora in evoluzione: le tecnologie risentono di continui perfezionamenti, ma sono soprattutto i metodi ad essere sottoposti ad esperimenti e a segnare continui progressi.
Qualcosa, e di molto positivo, è stato realizzato ma tutto sommato crediamo che il più e il meglio siano ancora da fare in questo campo.
Tutto quanto abbiamo visto finora di fatto ha un carattere abbastanza frammentario, episodico, anche se in certi casi viene applicato in modo sistematico.
Noi crediamo che il ruolo dell'informatica nella didattica sia ancora più importante e decisivo e investa la gestione della classe nel suo complesso.
Certo, gestire una classe scolastica è un'arte difficile, un'arte che richiede nell'insegnante parecchie doti, non solo abilità didattica: è capacità di governo, cioè di stimolare chi fa fatica, di suscitare interesse per la materia, di valutare come e

quando intervenire con richiami o incoraggiamenti.
Soprattutto è l'arte di chiamare gli allievi ad una partecipazione attiva tanto nei momenti in cui i contenuti vengono proposti quanto nei momenti in cui vengono valutati i risultati raggiunti.
Ebbene noi crediamo che l'informatica possa offrire degli strumenti assolutamente originali a servizio della gestione della classe.
Strumenti che non sostituiscono certamente il ruolo dell'insegnante, ma che gli permettono di svolgere il suo compito in modo totalmente nuovo e in certo senso rivoluzionario rispetto alla visione tradizionale.
Intendiamo dire che la presenza di un elaboratore in una classe può servire ad una gestione organica di tutto il processo di apprendimento, fungendo contemporaneamente da registro di classe, da archivio e da documentazione del lavoro svolto, e fornendo gli opportuni strumenti per una valutazione adeguata e costante dei progressi e in generale del cammino dei singoli e dell'intera classe.
Non si tratta evidentemente di cose nuove: più o meno sono aspetti che abbiamo già esaminato, e che si stanno realizzando. Si tratta piuttosto di un modo nuovo di gestire la classe: cioè tutti gli aspetti visti vanno integrati in una visione d'insieme. Vediamo singolarmente questi vari aspetti.

Il problema dell'autovalutazione

La valutazione è un momento fondamentale nel processo didattico ed abbiamo visto nella lezione precedente come l'elaboratore può essere un valido strumento in questo compito e come possa essere utilizzato a vari livelli.
Ma un aspetto ancora più importante è l'uso dell'elaboratore da parte degli allievi per un'autovalutazione.
La meccanizzazione del processo valutativo non è soltanto un alleggerimento del lavoro dell'insegnante ed un potenziamento delle sue capacità.
È anche, e a questo punto diremmo soprattutto, un processo di conduzione degli allievi verso una partecipazione più diretta e responsabile al problema della valutazione.

Alcuni esempi di utilizzo di elaboratore in una classe: nella pagina precedente, lezione assistita da elaboratore con vista su schermo gigante. In questa pagina, lezione su Van Gogh per mezzo di videodisco gestito da un elaboratore.



Tutti sappiamo che il docente non è infallibile e che la sua valutazione risente spesso dei limiti che gli sono inevitabilmente connessi. Una valutazione automatica può contribuire a ridurre questi difetti.
Ma, cosa ancor più importante, la valutazione automatica induce un nuovo modo di considerare l'errore: sembra infatti molto più sopportabile e meno traumatico per l'allievo l'errore fatto rilevare da una macchina, in un rapporto riservato, e magari con la possibilità di correggerlo mediante la presentazione, da parte dell'elaboratore, del materiale per il relativo recupero. È tutta la "filosofia dell'errore" che cambia. Le prove d'esame non dovrebbero diventare più dei momenti di tensione, dai quali dipende l'esito di un lungo periodo di apprendimento. La frequenza delle prove dovrebbe garantire agli allievi un controllo costante del procedere dell'apprendimento, e quindi il conseguimento diremmo naturale del titolo a cui il corso dà diritto.
Ciò non significa che il docente venga escluso da questo processo, ma piuttosto che debba seguire e dirigere a distanza il tutto, diradando gli interventi diretti.
Senza contare poi che il docente potrebbe (e dovrebbe) aver preparato parzialmente o totalmente i programmi che effettuano la valutazione e quindi sarebbe certamente coinvolto anche in altro modo al processo.
Ma, insistiamo, la valutazione meccanica generalizzata dovrebbe togliere la drammaticità dell'errore e della relativa correzione e quindi dovrebbe assicurare una partecipazione più attiva, responsabile e meno traumatica dello studente al processo di valutazione.

Registrazione e documentazione del lavoro svolto dalla classe e dai singoli

Il problema della valutazione, su cui s'innesta l'aspetto dell'autovalutazione, che abbiamo esaminato, non è che una parte di un problema ancora più vasto, più complesso, cui l'informatica può fornire gli strumenti per la risoluzione: il problema della documentazione del lavoro che è stato svolto dalla classe e dai singoli.

Supponiamo che in una ipotetica classe esista la possibilità di usare molto materiale C.A.I. (Istruzione Assistita da Calcolatore) sotto varie forme. Facciamo l'ipotesi che ad esempio almeno la metà delle lezioni e quasi tutte le valutazioni si svolgano in questa forma.
È abbastanza facile prevedere la possibilità di una documentazione generalizzata attraverso un sistema di elaborazione, di tutto il lavoro svolto e dei risultati raggiunti.
E questa documentazione sarebbe accessibile a tutti, docenti e alunni. Tutti potrebbero, in ogni momento, rendersi conto del cammino fatto, delle tappe superate, dei risultati effettivi confrontati con quelli previsti, e del cammino che di volta in volta resta ancora da fare.
Si potrebbero controllare i tempi e le velocità di apprendimento e tutti sarebbero più direttamente coinvolti anche per quanto concerne le decisioni da prendere nei vari momenti.
Una tale documentazione potrebbe tranquillamente e ampiamente sostituire il registro di classe e favorirebbe certamente il coinvolgimento degli studenti in tutte le fasi del processo didattico.
Quanto alle macchine che possono gestire tutto il processo è chiaro che dovrebbero superare le capacità del personal, a meno che i progressi della tecnologia portino rapidamente a produrre dei personal molto più potenti degli attuali.
È quindi opportuno prevedere la presenza di un minielaboratore che controlli il complesso del lavoro, attraverso tanti personal collegati a una notevole memoria di massa che garantisca la documentazione complessiva.
Gli studenti continuerebbero come adesso a lavorare sui personal, ma tutta l'attività verrebbe raccolta opportunamente attraverso l'elaboratore centrale, nella memoria di massa a cui fare riferimento tutte le volte che si vuole avere una documentazione completa del lavoro svolto dai singoli o dal complesso della classe.

Problemi

Diciamo chiaramente che quanto esposto rappresenta soltanto una serie di ipotesi per un lavoro futuro, poiché allo stato attuale non esistono esperimenti completi nel senso esaminato, ma soltanto tentativi più o meno frammentari.
Di fronte a queste possibilità vogliamo esaminare in dettaglio i problemi che si presentano.
Anzitutto ci si domanda qual è il costo delle apparecchiature e del software necessario. Per le apparecchiature i costi non dovrebbero essere proibitivi, anche se certo superiori a quelli del materiale attualmente usato che in genere consiste soltanto di elaboratori di tipo personal.
I personal da usare in questa nuova prospettiva dovrebbero essere collegati con un minielaboratore che ne controlla il processo. E poi ci sarebbe il costo del minielaboratore con una notevole memoria di massa.
L'impegno maggiore comunque dovrebbe essere rivolto al software, perchè non esiste attualmente un tipo che svolga i compiti visti, che sono completamente nuovi.
E si tratta di compiti certamente non facili, da sottoporre a tentativi, esperimenti, prove.
Si dovrebbe procedere qui, come in tanti altri campi, per prove ed errori, accumulando e sommando le esperienze positive e correggendo quelle negative.
Un secondo problema è il ruolo che verrebbe ad assumere l'insegnante in questo processo.
Anzitutto sarebbe chiamato a collaborare alla preparazione o almeno all'adattamento del software necessario, ma soprattutto sarebbe chiamato a "presiedere" a tutto il lavoro, controllando e intervenendo dove e come e quando si rende necessario.
È chiaro che non è possibile pensare ad un insegnamento totalmente svolto mediante l'elaboratore. Sarebbe certamente frustrante e riduttivo. Pensiamo piuttosto ad una presenza massiccia dell'elaboratore alternata a momenti di lezioni in aula (assistite da elaboratore attraverso quello che abbiamo chiamato "C.A.I. al docente"). Non mancherebbero poi i momenti di valutazione con intervento diretto del docente.
Ampio spazio dovrebbe essere lasciato anche alla discussione, agli scambi di esperienze tra allievi, alla valutazione collettiva del procedere del cammino didattico.
Resta infine il problema principale: come reagiranno gli studenti?
Certo, se uno guarda alle cose dall'esterno, si potrebbe anche spaventare nell'immaginare un insegnamento puramente meccanico, affidato a macchine onnipotenti e anonime.
Niente di tutto questo: si dovranno prevedere, come si diceva, momenti di lezione comune, di valutazione diretta da parte dell'insegnante, di discussioni collegiali sull'andamento del processo nel suo complesso.
Ripetiamo, lo scopo finale resta quello di aumentare e di migliorare la partecipazione e noi crediamo che gli strumenti indicati, opportunamente adattati, possano fornire la via.

Conclusioni

Non resta molto da aggiungere a quanto è stato detto.
Siamo partiti da considerazioni generali sul ruolo della didattica e principalmente sul ruolo che ha il problema nella didattica.
Crediamo che quanto presentato consenta di provare l'affermazione fatta che l'informatica può offrire strumenti e metodi nuovi e originali per l'impostazione e la soluzione dei problemi veri.
Certo, tutto sommato l'informatica ha una storia recente, si può dire che è ancora alle prime armi, nonostante i rapidi progressi tecnologici.
A maggior ragione possiamo affermare che anche la didattica è ancora alle prime armi per quanto riguarda l'applicazione di strumenti e metodi informatici.
Eppure guardiamo al futuro con grande fiducia: dovranno essere fatti nuovi tentativi, nuove esperienze, nuove prove; si dovrà tener conto di risultati positivi e negativi, si dovranno continuamente modificare metodi e strumenti... ma siamo profondamente convinti che i risultati, alla fine, daranno ragione al nostro fondamentale ottimismo.

IL LINGUAGGIO COBOL (I)

Un linguaggio di programmazione estremamente ricco e soprattutto adatto alla soluzione di problemi di natura commerciale o amministrativa.

Cenni storici

Intorno alla fine degli anni Cinquanta il governo degli Stati Uniti incaricò un'organizzazione di nome CODASYL (COnference on DAta SYstem Language) di definire un linguaggio adatto alla programmazione di applicazioni amministrative e gestionali, di applicazioni, cioè, non particolarmente difficili dal punto di vista algoritmico, ma notevolmente complesse dal punto di vista della grande quantità di dati da elaborare e delle loro interrelazioni.
Oggi esistono numerose versioni di questo linguaggio: la versione COBOL ANSI (1974) è considerata lo standard.

Caratteristiche generali

Come al solito, prima di dare indicazioni specifiche sul linguaggio, esaminiamo un programma per vedere come questo si presenta.
Si tratta di un esempio molto semplice, che legge un listino prezzi e ne permette una reiterata interrogazione, fornendo il prezzo corrispondente a ciascun elemento trovato; l'esempio è piuttosto rozzo dal punto di vista delle funzioni svolte, essendo il suo obiettivo quello di evidenziare le caratteristiche del linguaggio.
Un programma COBOL è costituito da quattro *divisioni*:

```
000001 IDENTIFICATION DIVISION.          RICERCA.
000002 PROGRAM-ID.                       ETNOTEAM.
000003*AUTORE:                            MAGGIO 84.
000004*DATA DI PROGETTO:
000005*
000006* ************************************************************
000007 ENVIRONMENT DIVISION.
000008 CONFIGURATION SECTION.            DPS-4.
000009 SOURCE-COMPUTER.                  DPS-4.
000010 OBJECT-COMPUTER.
000011* ************************************************************
000012 DATA DIVISION.
000013 WORKING-STORAGE SECTION.
000014*
000015 01  LISTINO.                     OCCURS 100.
000016     02   ELEM-LISTINO            PIC X(10).
000017          03   NOME               PIC 9(9).
000018          03   PREZZO
000019*
000020 01  WORK-AREA.                   PIC 9(2).
000021     02   IND                     PIC X(10).      VALUE "N".
000022     02   ELEM                    PIC X
000023     02   TROVATO                 PIC X.
000024     02   RISP                    PIC X(10).
000025     02   NOME-IND                PIC 9(9).
000026     02   PREZZO-IND              PIC 9(3).
000027     02   NUM-ELEM
000028* ************************************************************
```

▶ segue

```
000029 PROCEDURE DIVISION.
000030 MAIN-PROGRAM.
000031      DISPLAY "* QUANTI ELEMENTI ?"
000032      ACCEPT NUM-ELEM.
000033      PERFORM  LEGGI-LISTA
000034      THRU END-LEGGI-LISTA
000035           VARYING IND FROM 1 BY 1
000036           UNTIL   IND > NUM-ELEM.
000037*
000038      DISPLAY "* RICERCA (S/N) ?"
000039      ACCEPT RISP.
000040*
000041      PERFORM  RICERCA
000042      THRU END-RICERCA
000043           UNTIL RISP = "N".
000044*
000045 FINE-PROGRAM.
000046      STOP RUN.
000047*     ***************************************************************
000048 LEGGI-LISTA.
000049      DISPLAY "* INSERIRE NOME E PREZZO *"
000050      ACCEPT  NOME-IND
000051      MOVE    NOME-IND              TO NOME(IND)
000052      ACCEPT  PREZZO-IND
000053      MOVE    PREZZO-IND            TO PREZZO(IND).
000054*
000055 END-LEGGI-LISTA.
000056*     ***************************************************************
000057 RICERCA.
000058      DISPLAY "* NOME ?"
000059      ACCEPT NOME-IND.
000060           MOVE "N"                 TO TROVATO.
000061      PERFORM  CERCA
000062      THRU END-CERCA
000063           VARYING IND FROM 1 BY 1
000064           UNTIL   TROVATO = "S"    OR
000065                   IND     > NUM ELEM.
000066*
000067      IF TROVATO  = "S"
000068           DISPLAY "* PREZZO = " PREZZO(IND).
000069*
000070      DISPLAY "* RICERCA (S/N) ?"
000071      ACCEPT RISP.
000072*
000073 END-RICERCA.
000074*     ***************************************************************
000075 CERCA.
000076      IF NOME-IND = NOME(IND)
000077           MOVE "S"                 TO TROVATO.
000078 END-CERCA.
```

- la IDENTIFICATION DIVISION (Divisione Identificazione, righe 1-6), ha lo scopo di identificare il programma e la sua collocazione nell'ambiente di sviluppo;

- la ENVIRONMENT DIVISION (Divisione Ambientamento, righe 7-11) ha lo scopo di indicare quale ambiente di elaborazione viene usato;

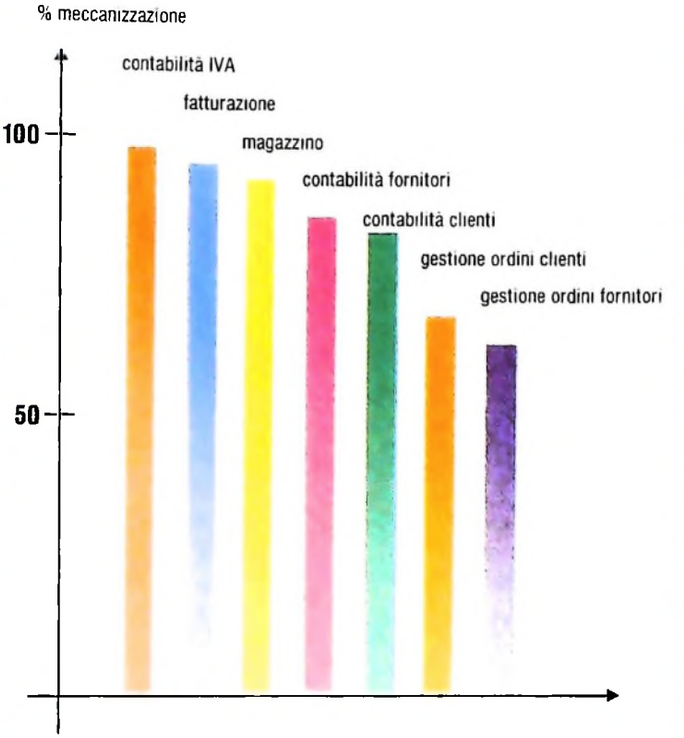
- la DATA DIVISION (Divisione Dati, righe 12-28) raccoglie le definizioni dei file e delle variabili;

- la PROCEDURE DIVISION (Divisione Procedura, righe 29-78) contiene le istruzioni del programma.

Nella parte di IDENTIFICATION DIVISION è stato introdotto il nome del programma (RICERCA) che segue la locuzione PROGRAM-ID., il nome dell'autore e la data di progetto sotto forma di commenti; questi ultimi sono identificati dall'asterisco (righe 4 e 5). Autore e data sono esprimibili anche con frasi COBOL con lo stesso significato.
La ENVIRONMENT DIVISION, che potrebbe contenere numerose informazioni, è qui usata per dichiarare il SOURCE-COMPUTER (riga 9), ovvero qual è il calcolatore su cui il programma viene sviluppato (in questo caso il calcolatore Honeywell DPS4) e l'OBJECT-COMPUTER, ovvero qual è il calcolatore su cui dovranno essere eseguiti i programmi tradotti dal compilatore. In questo caso i due ambienti coincidono, ma a priori è pensabile che un compilatore funzionante su un calcolatore produca una traduzione per un altro (in questo caso si parla di CROSS COMPILER).
La DATA DIVISION che in generale contiene la descrizione di tutti i file usati dal programma e la definizione di tutte le variabili di memoria, in questo esempio si limita alle variabili che sono descritte nella cosiddetta WORKING-STORAGE SECTION.
Le dichiarazioni di variabili COBOL sono orientate alla costruzione di record, cioè di informazioni strutturate che contengono diverse variabili di tipo differente; nell'esempio vediamo la dichiarazione di una variabile LISTINO (riga 15), di livello 01, ovvero al massimo livello di astrazione, con la possibilità di specificarne la struttura. Tale variabile viene infatti ulteriormente specificata a un livello 02, ove si dice che LISTINO è composta da un elemento ELEM-LISTINO, che compare 100 volte (riga 16): si tratta cioè di un array di 100 elementi.
Tale livello 02 viene ulteriormente raffinato in due campi di livello 03, il primo identificato da NOME e caratterizzato dal tipo, una stringa di 10 caratteri (evidenziata dalla definizione PIC X (10)), il secondo identificato da PREZZO e specificato di tipo numerico di 9 cifre (la definizione PIC 9(9)).
Il programmatore ha poi raccolto tutte le aree di lavoro in un'ulteriore struttura identificata dal nome WORK-AREA, ulteriormente suddivisa nelle singole variabili.
Si notino la variabile IND (numerica di 2 cifre), la variabile TROVATO di un carattere: quest'ultima ha un valore iniziale (cioè inserito direttamente dal compilatore) pari al carattere "N" (riga 23).
La PROCEDURE DIVISION è organizzata in PARAGRAFI.

Le meccanizzazioni tradizionali

Le meccanizzazioni classiche sono tradizionalmente realizzate in linguaggio Cobol.



essendo un paragrafo un insieme di istruzioni compreso tra due "etichette" (come MAIN-PROGRAM alla riga 30 o LEGGI-LISTA alla riga 48).
Il primo paragrafo, da MAIN-PROGRAM (riga 30) a FINE-PROGRAM (riga 45) effettua le operazioni di:

- visualizzazione di un messaggio con l'istruzione DISPLAY (riga 31);

- lettura della variabile NUM-ELEM (riga 32);

- esecuzione del paragrafo compreso tra le etichette LEGGI-LISTA ed END-LEGGI-LISTA (righe 33 e 34) facendo variare la variabile IND dal valore iniziale 1, con incrementi di 1 (riga 35), fino al momento in cui IND è maggiore di NUM-ELEM (riga 36).

Il paragrafo successivo contiene solo l'istruzione STOP RUN (riga 46), che richiede la terminazione del programma.
L'istruzione PERFORM causa l'esecuzione del paragrafo indicato, alla fine del quale il controllo ritorna al punto della chiamata, con la possibilità delle eventuali ripetizioni.

Si noti la possibilità di definire numerose condizioni di uscita in una iterazione, come per esempio nelle linee 61-65, ove l'iterazione enumerativa è affiancata dalla terminazione per elemento trovato.
Si osservi la forma dell'assegnamento nella riga 51 o nella riga 77, con la parola chiave MOVE.
Si noti infine la presenza del "." come elemento di terminazione di una sequenza di operazioni.
Il COBOL si presenta quindi come un linguaggio verboso, che tende ad avvicinare il programma a una descrizione vicina alla lingua inglese.

Astrazione nella definizione di dati

La DATA DIVISION, che abbiamo esaminato in una versione molto ridotta, può presentare numerose SEZIONI con differenti tipi di informazioni:

- la FILE-SECTION è legata alla descrizione dei file, e permette di definirne le caratteristiche logiche (tipo di accesso, aree di lettura associate...) e fisiche (modalità di registrazione su disco...); aree di memoria associate all'uso di file sono definite come in WORKING-STORAGE SECTION;

- la WORKING-STORAGE SECTION, che permette di descrivere le variabili interne al programma e di definire strutture a diversi livelli di raffinamento, mettendo a disposizione strutture di tipo array e record; i tipi elementari disponibili sono costituiti fondamentalmente da CARATTERI (PIC X), da caratteri ALFABETICI (PIC A), da INTERI (PIC 9), da numeri dotati di parte decimale, da variabili che (non entriamo qui in dettagli troppo tecnici) possono essere utilizzati per simulare dati BOOLEAN;

- la LINKAGE-SECTION specifica informazioni legate a parametri che debbano essere passati da un programma a un altro;

- la COMMUNICATION-SECTION consente di definire aree di memoria da usare per la comunicazione con terminali;

- la REPORT-SECTION permette di definire la struttura di tabulati di stampa da produrre, allo scopo di usare particolari procedure automatiche per la costruzione dei rapporti.

Dal punto di vista delle caratteristiche di astrazione il COBOL presenta quindi aspetti estremamente avanzati insieme ad aspetti che evidenziano la sua data di nascita: accanto alla disponibilità di array e di record con la possibilità di definizione di dati a diversi livelli di astrazione, sono presenti tipi di dati elementari poveri.
Particolare interesse rivela invece la REPORT-SECTION, che si configura come un vero e proprio aspetto di LINGUAGGIO NON PROCEDURALE.
Si definiscono PROCEDURALI i linguaggi di programmazione che richiedono la descrizione di tutte le operazioni da fare per effettuare una certa elaborazione, con l'ordine di esecuzione (tipicamente il BASIC, il FORTRAN); si dicono linguaggi NON PROCEDURALI quelli in cui si definisce invece l'obiettivo da raggiungere, ma non il modo di raggiungerlo; questo atteggiamento, molto avanzato, è legato evidentemente alla crescita di astrazione in un linguaggio: l'esecuzione di un assegnamento con una complessa espressione aritmetica del FORTRAN è visto come "non procedurale" se esaminato dal punto di vista dell'Assembler, che deve invece specificare passo passo tutte le operazioni da effettuare.
Il livello di astrazione permesso dalla REPORT-SECTION è altissimo; essa definisce infatti la struttura del rapporto di stampa in termini di righe e colonne:

- dimensioni fisiche della pagina di stampa
- presenza di un'intestazione del rapporto
- presenza di un'intestazione di ogni singola pagina
- numerazione delle pagine
- posizioni della prima e dell'ultima riga del foglio
- contenuti delle varie righe, opportunamente classificate e così via.

Opportuni verbi del tipo INITIATE, TERMINATE, GENERATE permettono di costruire il rapporto senza più assolutamente preoccuparsi della sua struttura fisica o del raggiungimento di fine pagina; o della numerazione delle pagine.
Il COBOL è un linguaggio ad allocazione statica; quindi tutte le strutture di dati definite nella DATA DIVISION sono predisposte dal compilatore e sempre presenti in memoria. Esse sono visibili da ogni punto della PROCEDURE DIVISION.
È tuttavia possibile ridefinire una stessa area con nomi diversi mediante un'adeguata clausola REDEFINES.
Per esempio,

```
01 ALFA.
   02 X PIC 9.
   02 Y PIC X.
01 BETA REDEFINES ALFA.
   02 Z PIC XX.
```

definisce un'area ALFA composta da due variabili X e Y rispettivamente numerica e alfanumerica, BETA è un nome della stessa area di memoria, vista però come un'unica variabile alfanumerica di 2 caratteri.
Tale possibilità è particolarmente significativa nel caso in cui si debbano leggere informazioni da un file esterno, che possono essere organizzate in modo differente. In tali casi spesso si usa una ridefinizione della stessa area, mantenendo un campo iniziale comune a tutte le definizioni, da usare come indicatore del tipo di record. Una volta letto il record nell'area, l'analisi del valore di tale campo permette l'adozione della ridefinizione adeguata; si tratta in sostanza di meccanismi che si ritrovano, con una maggior pulizia formale, nei cosiddetti "record con varianti" del linguaggio PASCAL.

Anche in leasing con Olivetti Leasing

LA FAMIGLIA DEI PERSONAL COMPUTER OLIVETTI

FRIENDLY & COMPATIBLE

Questa famiglia di personal compatibili tra loro e con i più diffusi standard internazionali, non ha rivali per espandibilità e flessibilità. Prestazioni che su altri diventano opzionali, sui personal computer Olivetti sono di serie. Per esempio M24 offre uno schermo ad alta definizione grafica, ricco di 16 toni o di 16 colori e con una risoluzione di 600x400 pixel; mentre la sua unità base dispone di 7 slots di espansione, fatto questo che gli consente di accettare schede di espansione standard anche se utilizza un microprocessore a 16 bit reali (INTEL 8086).
Ma ricchi vantaggi offrono anche tutti gli altri modelli.

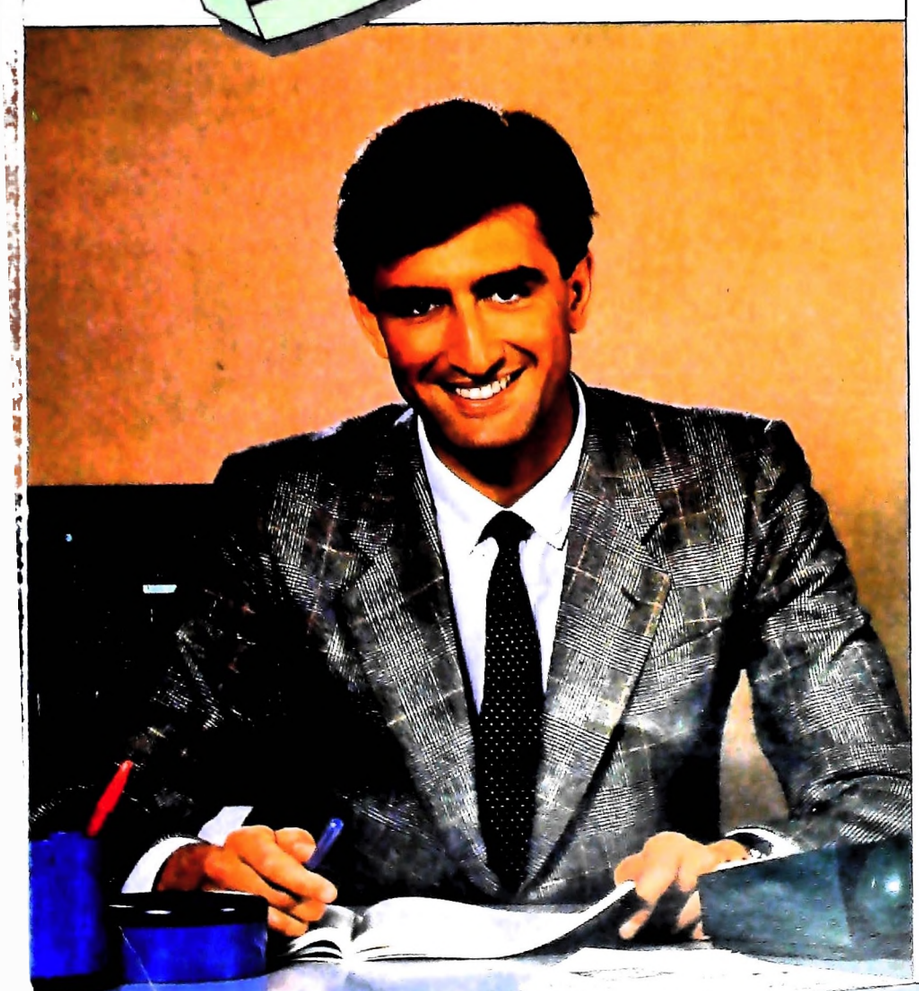
Basti pensare che tutte le unità base includono sia l'interfaccia seriale che quella parallela. Oppure basti pensare all'ampia gamma di supporti magnetici: floppy da 360 a 720 KB o un'unità hard disk (incorporata o esterna) da 10 MB.
La loro compatibilità, inoltre, fa sì che si possa far uso di una grande varietà di software disponibile sul mercato. Come, ad esempio, la libreria PCOS utilizzabile anche su M24. Come le librerie MS-DOS®, CP/M-86® e UCSD-P System®, utilizzabili sia da M20 che da M21 e M24.

MS-DOS è un marchio Microsoft Corporation
CP/M-86 è un marchio Digital Research Inc.
UCSD-P System è un marchio
Regents of the University of California

olivetti

— UN NUOVO MODO DI USARE LA BANCA.—

Consulenza



GLI INVESTIMENTI CON VOI E PER VOI DEL BANCO DI ROMA.

Il Banco di Roma non si limita a custodire i vostri risparmi. Vi aiuta anche a farli meglio fruttare. Come? Mettendovi a disposizione tecnici e analisti in grado di offrirvi una consulenza di prim'ordine e di consigliarvi le forme di investimento piú giuste. Dai certificati di deposito ai titoli di stato, dalle obbligazioni alle azioni, il Banco di Roma vi propone professionalmente le varie opportunità del mercato finanziario. E grazie ai suoi "borsini", vi permette anche di seguire, su speciali video, l'andamento della Borsa minuto per minuto.

Se desiderate avvalervi di una gestione qualificata per investire sui più importanti mercati mobiliari del mondo, i fondi comuni del Banco di Roma, per titoli italiani ed esteri, vi garantiscono una ampia diversificazione.

Inoltre le nostre consociate Figeroma e Finroma forniscono consulenze per una gestione personalizzata del portafoglio e per ogni altra esigenza di carattere finanziario.

Veniteci a trovare, ci conosceremo meglio.

BANCO DI ROMA
CONOSCIAMOCI MEGLIO.