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

# 31 CORSO PRATICO COL COMPUTER

421864

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

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 (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, ETTORE DECIO, GIANCARLO MAURI, Eidos (TIZIANO BRUGNETTI), Etnoteam (ADRIANA BICEGO)

Tavole
Logical Studio Communication
Il Corso di Programmazione e BASIC è stato realizzato da Etnoteam S.p.A., Milano
Computergralica 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 BREGGÈ

Segretarie di Redazione
RENATA FRIGOLI
LUCIA MONTANARI

# INDEX (REGISTRO INDICE)

## Con l'introduzione del registro INDEX i microprocessori si evolvono e cambiano la loro architettura: analizziamo nel dettaglio le diverse possibilità di applicazione.

I primi microprocessori erano stati progettati per risolvere problemi tecnici, in special modo quelli di controllo. Le loro funzioni erano concepite per questi problemi ed era compito degli specialisti che usavano questi elementi provvedere a tutto il necessario perché essi potessero funzionare correttamente. Man mano che crescevano le applicazioni, sia nel campo tecnico sia in quello amministrativo, i microprocessori si arricchirono di nuove funzioni e incominciarono ad avere architetture che resero più semplice la loro utilizzazione da parte degli utenti. Inoltre, mentre inizialmente il software era subordinato all'hardware, successivamente le cose incominciarono a cambiare e giustamente le necessità del software incominciarono a condizionare l'architettura dei microprocessori.

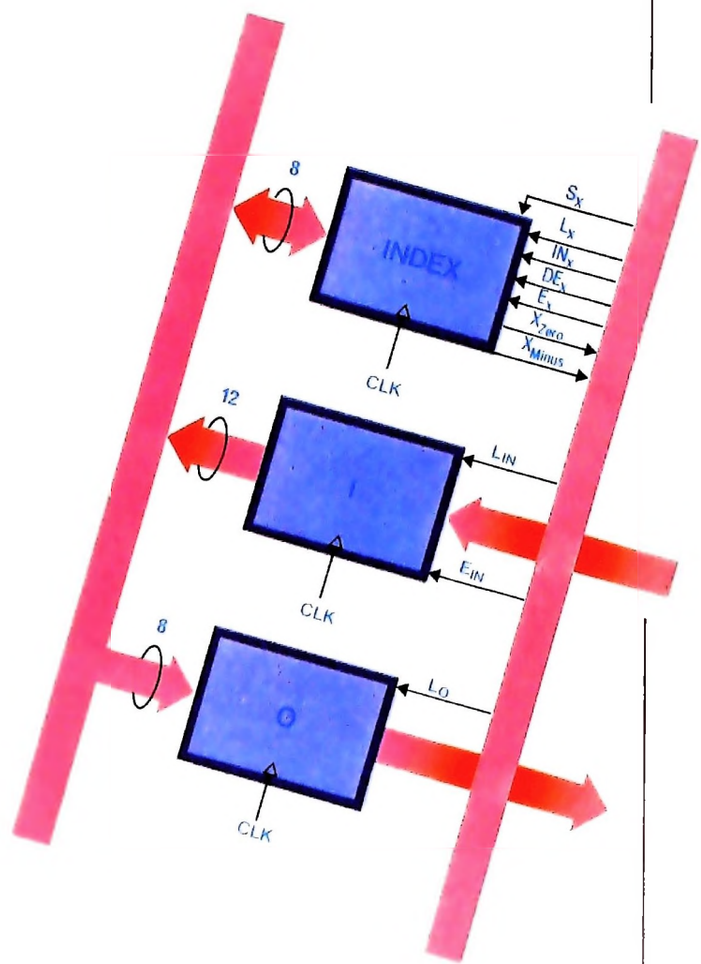
Il registro Index è uno dei prodotti di questo evolversi dei microprocessori. I primi ne avevano uno, come il 6800 per esempio, poi vennero quelli con due, come lo Z80 e altri; attualmente i microprocessori moderni ne contengono una quantità considerevole. Ma vediamo nel dettaglio cosa si può fare con questo registro (figura a lato).

*Registro ausiliario*: cella di memoria interna rapida, accessibile dall'utente, molto utile per il deposito momentaneo di variabili del programma.

*Registro contatore*: è possibile, per esempio, caricare un numero e decrementarlo con il comando $DE_X$ ogni volta che accade un evento in modo che, quando il contenuto diventa zero, avremo un segnale nella linea XZero che indica il termine del conteggio. È possibile anche incrementare il suo contenuto con il comando $IN_X$, confrontarlo con un determinato valore e agire di conseguenza quando si verifica l'uguaglianza.

*Registro indice*: uno degli usi più significativi è proprio quello di indice. Prendiamo per esempio l'indirizzo del primo elemento di una lista che si trova in memoria e depositiamolo nel registro INDEX. In queste condizioni è particolarmente facile prelevare tutti i valori della lista incrementando soltanto il contenuto del registro INDEX.

*Indice + Spostamento*: questa funzione, importantissima per il software, permette di ricollocare interi programmi, cioè di spostarli da una parte all'altra della memoria senza perdere la fun-




Schema di funzionamento di un registro INDEX, oggi applicato a tutti i microprocessori in notevole quantità.

zionalità. In definitiva l'indirizzo dove si va a prelevare o depositare i dati è dato dalla somma del contenuto del registro INDEX più uno spostamento.
I due comandi $IN_X$ e $DE_X$ incrementano e decrementano il contenuto del registro INDEX alla transizione del clock, mentre XMinus e XZero indicano rispettivamente che il contenuto è negativo o zero.
Con il comando $S_X = 1$ si somma il valore XX all'INDEX.
Con $L_X$ si carica, mentre con $E_X$ si abilita il registro INDEX.

## "I" (registro di entrata)

Questo registro è un circuito di interfaccia che raccoglie i dati provenienti dagli elementi periferici, per esempio tastiera, nastro magnetico ecc., per passarli ai registri interni. Quando il comando, $L_{IN} = 1$ alla transizione positiva del clock, il dato esterno viene caricato nel registro. Con $E_{IN} = 1$ i dati passano al BUS.

## "O" (registro di uscita)

Registro con azione inversa al registro "I". Con $L_O = 1$ alla transizione positiva del clock si carica il dato nel registro e subito viene presentato all'esterno.

## Istruzioni MRI

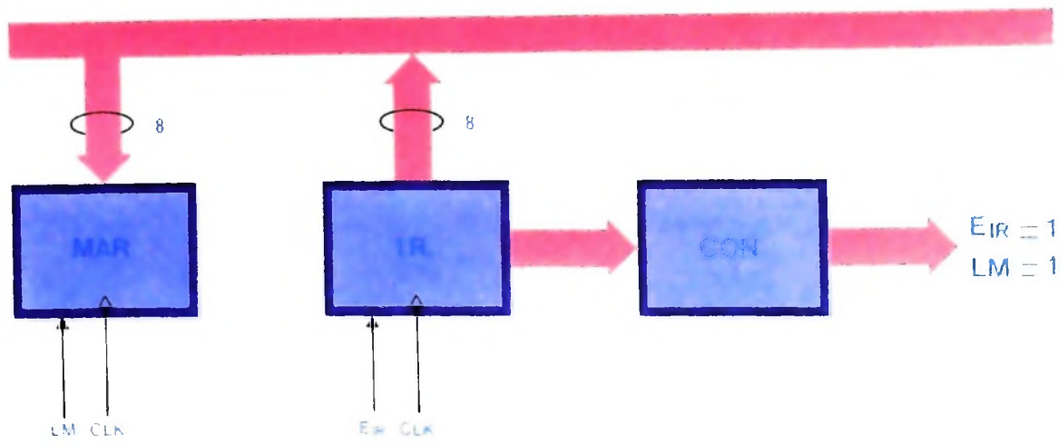
La Uamicro II possiede 32 istruzioni, 12 sono MRI. Naturalmente la quantità di istruzioni è condizionata dall'architettura che scegliamo; in questo caso, per esempio, abbiamo definito una parola di 12 bit: 4 per il codice di istruzioni, quindi al massimo 16 MRI, e 8 bit per l'operando, cioè 256 indirizzi di memoria. Se avessimo scelto una parola di 16 bit, avremmo potuto dividerla in due parti, per esempio, 5 per il codice di istruzione (con 32 valori diversi) e 11 per l'operando quindi fino a 2048 indirizzi di memoria.
Le istruzioni che restano, cioè quelle non MRI, sono istruzioni riguardanti le operazioni sui registri interni e queste possono essere molte di più (256 in totale) perché possono essere selezionate con tutti gli 8 bit dell'operando.

## Fetch (ciclo di ricerca)

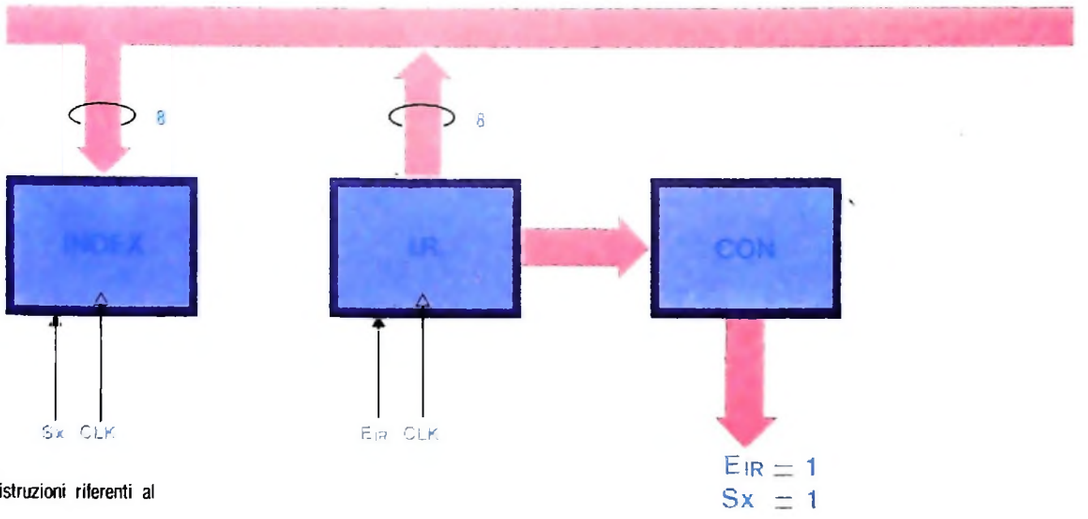
Il ciclo di ricerca *fetch* è uguale, in definitiva, a quello della Uamicro I:
To = (PC o SC) → MAR; fase di indirizzo
Ti = MEM → IR; ((PC o SC) + 1) fase di memoria e incremento del PC o SC.
Come già sappiamo tutte le istruzioni passano per IR per essere decodificate per poi inviare la giusta informazione al CON. Se l'istruzione è MRI gli 8 bit dell'operando (figura in basso) ritornano al BUS come indirizzo dove vengono catturate dal MAR e quindi ripresenti alla memoria. Se invece sono di scostamento vanno sommati al contenuto dell'INDEX (figura in alto, pagina seguente) e il valore così ottenuto costituisce l'indirizzo da caricare nel MAR.
Vediamo quali sono le istruzioni:

| | |
|---|---|
| LDA XX | A ← (XX) |
| ADD XX | A ← (A + (XX)) * |
| SUB XX | A ← (A - (XX)) * |
| STA XX | (XX) ← A |
| LDB XX | B ← (XX) |
| LXA XX | A ← (INDEX + XX) * |
| SXA XX | (INDEX + XX) ← A * |

* istruzioni MRI composte.




Configurazioni delle istruzioni MRI.




Configurazioni delle istruzioni riferenti al registro INDEX.

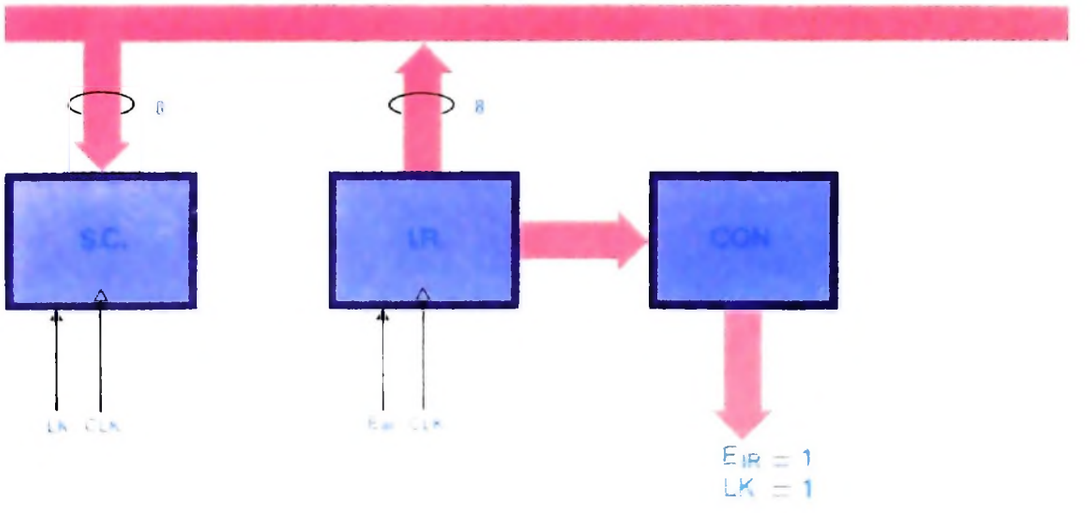
## Istruzioni di salto

Ci sono 5 istruzioni di salto che si comportano nel seguente modo: nella fase $T_2$ (figura in basso) il contatore emette i comandi $E_{IR}$ e $L_K$, con il primo il valore XX passa al BUS mentre con $L_K$, al clock, entra nel registro SC (in uno dei registri interni) cambiando la direzione esistente se la condizione risulta valida.

JAM XX salto all'indirizzo XX se l'accumulatore è negativo.
JAZ XX salto all'indirizzo XX se l'accumulatore è zero.
JIM XX salto all'indirizzo XX se l'INDEX è negativo.
JIZ XX salto all'indirizzo XX se l'INDEX è zero.
JMS XX salto al sottoprogramma con indirizzo XX.

## Istruzioni "Operate"

Le istruzioni "Operate", non usano la memoria e nemmeno modificano il contenuto del PC o SC, bensì agiscono sui dati già depositati internamente. È bene soffermarci un po' su queste istruzioni per sottolineare le differenze funzionali con le MRI.
Le MRI hanno in definitiva il compito specifico di trasferire dati tra gli elementi esterni e la CPU, mentre le istruzioni "Operate" vengono eseguite direttamente sui dati. È ovvio che queste ultime sono più numerose delle prime e sono anche quelle che presentano più differenze fra fabbricante e fabbricante, ognuno dei quali sviluppa di più alcuni tipi di istruzioni secondo i propri




Configurazione delle istruzioni di salto.

| MNEM | CODICI | | | |
|---|---|---|---|---|
| NOP | 1 1 1 1 | 0 0 0 0 | X X X X | NO OPERATE |
| CLA | 1 1 1 1 | 0 0 0 1 | X X X X | CANCELLA "A" |
| XCH | 1 1 1 1 | 0 0 1 0 | X X X X | INTERCAMBIA A ⇆ X |
| DEX | 1 1 1 1 | 0 0 1 1 | X X X X | DECREMENTA INDEX |
| INX | 1 1 1 1 | 0 1 0 0 | X X X X | INCREMENTA INDEX |
| CMA | 1 1 1 1 | 0 1 0 1 | X X X X | COMPLEMENTA "A" |
| CMB | 1 1 1 1 | 0 1 1 0 | X X X X | COMPLEMENTA "B" |
| IOR | 1 1 1 1 | 0 1 1 1 | X X X X | "A" OR "B" |
| AND | 1 1 1 1 | 1 0 0 0 | X X X X | "A" AND "B" |
| NOR | 1 1 1 1 | 1 0 0 1 | X X X X | "A" OR "B" |
| NAN | 1 1 1 1 | 1 0 1 0 | X X X X | "A" AND "B" |
| XOR | 1 1 1 1 | 1 0 1 1 | X X X X | "A" O "A" |
| RTS | 1 1 1 1 | 1 1 0 0 | X X X X | RITORNO DAL SOTTOPROGRAMMA |
| IMP | 1 1 1 1 | 1 1 0 1 | X X X X | ENTRATA |
| OUT | 1 1 1 1 | 1 1 1 0 | X X X X | USCITA |
| HLT | 1 1 1 1 | 1 1 1 1 | X X X X | STOP |

Tabella delle istruzioni "Operate" con i registri interni, i codici operativi e il significato di ogni operazione.

punti di vista, o le proprie necessità. Vediamo quali sono.

NOP: questa istruzione, in particolare, non fa niente. Il suo uso dipende molto dalle circostanze software o dalle architetture. Per esempio, talvolta si ha bisogno di un'azione temporizzatrice specialmente in programmi di controllo: un'istruzione come questa, che si può ripetere molte volte senza che provochi cambiamenti nei dati, risulta utile.

Il suo impiego in hardware invece riguarda particolarmente quei microprocessori, come lo Z80, che prevedono la funzione di rinfresco delle memorie dinamiche (di queste necessità avremo modo di parlarne in dettaglio più avanti). Questo tipo di memorie ha bisogno di un impulso di rinfresco a intervalli di tempo regolari e pertanto, quando il microprocessore è fermo per qualsiasi motivo, continua a eseguire le istruzioni di NOP abbinate con un'azione di rinfresco.

CLA A ← O: azzera l'accumulatore.

XCH A⇌ INDEX: permette lo scambio di dati tra l'accumulatore e il registro INDEX.

DEX INDEX ← (INDEX - 1): decrementa l'INDEX.

INX INDEX ← (INDEX + 1): incrementa l'INDEX.

CMA A ← A: complementa l'accumulatore A.

CMB B ← B: complementa l'accumulatore B.

IOR A ← (A OR B): la funzione OR fatta bit per bit tra A e B.

AND A ← (A AND B): la funzione AND fatta bit per bit tra A e B.

NOR A ← (A NOR B): la funzione NOR fatta bit per bit tra A e B.

NAND A ← (A NAND B): la funzione NAND fatta bit per bit tra A e B.

XOR A ← (A XOR B): la funzione XOR fatta bit per bit tra A e B.

RTS: questa istruzione è l'ultima di qualsiasi sottoprogramma e ci permette di ritornare con comando al registro precedente.

INP o OUT: queste due istruzioni sono le tipiche istruzioni dei microprocessori che prevedono circuiti di entrata e uscita di carattere specifico. Naturalmente possono presentare numerose varianti.

HLT: questa istruzione ferma l'esecuzione del programma.

Nella tabella qui sopra sono raggruppati i codici delle istruzioni. Da notare che i quattro bit più significativi (quelli a sinistra) sono tutti 1. mentre gli ultimi quattro XXXX sono a disposizione del lettore per ampliare la lista di operazioni possibili.

# I SISTEMI ESPERTI: I PRINCIPI DI BASE

## Insiemi di programmi in grado di raggiungere, in specifici campi, una notevole operatività.

Si è visto, in precedenti articoli di questa rubrica, che il problema originario che si erano posti gli studiosi di Intelligenza Artificiale, e cioè riprodurre artificialmente una entità "pensante", si scisse presto in un insieme di attività interagenti, ognuna rivolta a un particolare sottocampo dell'idea di partenza e già di per sé estremamente complessa. Alcuni sviluppi sono stati esaminati in queste pagine, a esempio quelli legati alla visione e alla comunicazione uomo-macchina. Esamineremo ora un campo dell'A.I. molto studiato in questi ultimi anni, quello dei *sistemi esperti*: esso è legato al primo punto della classificazione proposta a pagina 137, il *ragionamento*. Questo campo di ricerca, insieme a quello sull'apprendimento cui è strettamente legato, appare forse il più arduo tra quanti sinora affrontati.

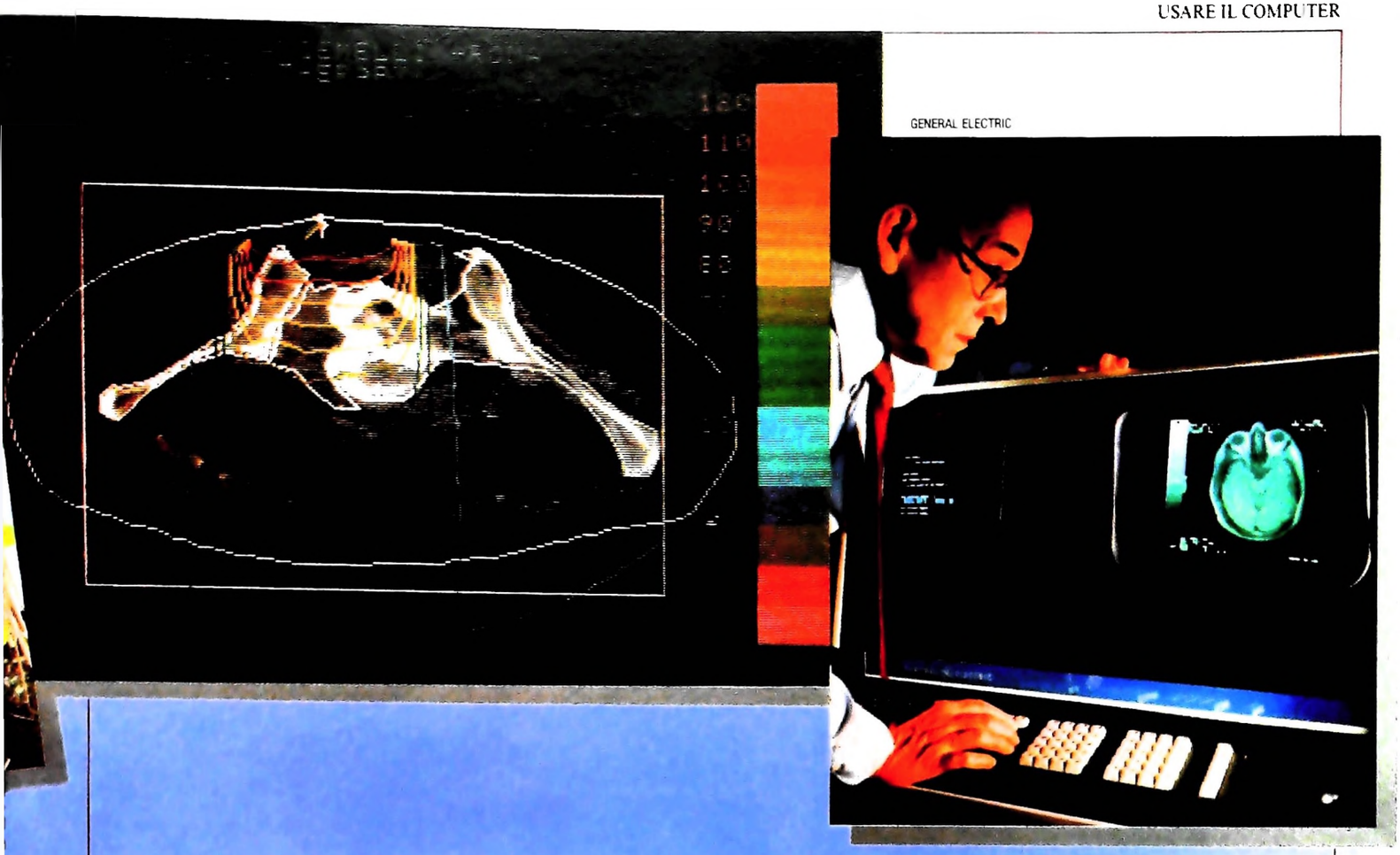
### Che cosa sono i sistemi esperti

I primi tentativi di dare una schematizzazione ai processi di ragionamento utilizzati dall'uomo nello svolgere un'attività intelligente risalgono praticamente ad Aristotele, e da allora sono stati spesi fiumi di inchiostro sull'argomento, secondo gli approcci più diversi. Non ci addentriamo in questi rischiosi meandri: è sufficiente che ciascuno rifletta e provi ad analizzare il vasto insieme di processi e attività mentali che vengono innescati, in modo del tutto automatico, nello svolgimento di un compito considerato anche relativamente semplice. Per fissare le idee, pensiamo di dover guidare un'automobile da Milano a Roma. Nel portare a termine questo compito interverranno diversi elementi, tra i quali:
1) capacità di astrazione: "per arrivare a Roma è necessario prendere l'autostrada del Sole, quindi l'attività è divisa in due parti: giungere al casello e percorrere il tratto autostradale; per ora non mi occupo dei dettagli relativi al modo di uscire dalla città";
2) ragionamenti probabilistici: "poiché è venerdì pomeriggio probabilmente il centro della città è affollato, quindi è bene non attraversarlo";
3) assunzioni: "l'automobile è a punto, i freni sono a posto, la batteria è carica, ...";
4) conoscenze specifiche: "si può andare da via Melzi d'Eril a piazza Firenze percorrendo corso Sempione";
5) conoscenze generali: "si guida a destra, ai semafori si parte con il verde, e così via".
Così come in altri campi la strada seguita in A.I. è stata quella di orientare la ricerca verso sottoproblemi singolarmente affrontabili, ben delimitati e specifici. L'attenzione è stata spostata dagli schemi di ragionamento globale, difficilmente formalizzabili in modo da poter essere tutti integrati in un unico sistema, alla conoscenza specifica e approfondita necessaria per la risoluzione di un problema in un preciso dominio ben conosciuto e studiato. A questo restringimento del campo ha d'altra parte corrisposto un incremento dell'utilità e delle prestazioni dei primi prototipi realizzati.
Sono nati così negli ultimi anni alcuni complessi insiemi di programmi denominati significativamente sistemi esperti. Loro scopo è di raggiungere, in particolari e specifici campi, un'operatività e funzionalità pari a quelle di un individuo esperto. Naturalmente i campi di applicazione, anche se precisi, non sono banali ma richiedono appunto una notevole esperienza e professionalità: per dare un'idea delle applicazioni citiamo per esempio la diagnosi di malattie infettive, la deduzione di strutture chimiche a partire da dati sperimentali, l'identificazione di giacimenti minerari, la configurazione e progettazione di sistemi di calcolo, il controllo intelligente di reattori nucleari e altri attualmente oggetto di studio sperimentale. Quella che è stata in origine una scelta obbligata, dovuta al disastroso fallimento nella realizzazione di sistemi che esibissero capacità di ragionamento generale su una vasta parte dello scibile umano, è stata alla lunga produttiva e ha dato origine a una vera e propria disciplina autonoma, l'*ingegneria della conoscenza*.
Il disporre di un esperto "artificiale" può avere diverse motivazioni:
1) il costo minore dell'interazione con un sistema computazionale rispetto a una consulenza umana;
2) la possibilità di interazione costante, anche quando un esperto può non essere disponibile;
3) in certi casi, la maggiore facilità di interazione esperto-utente.

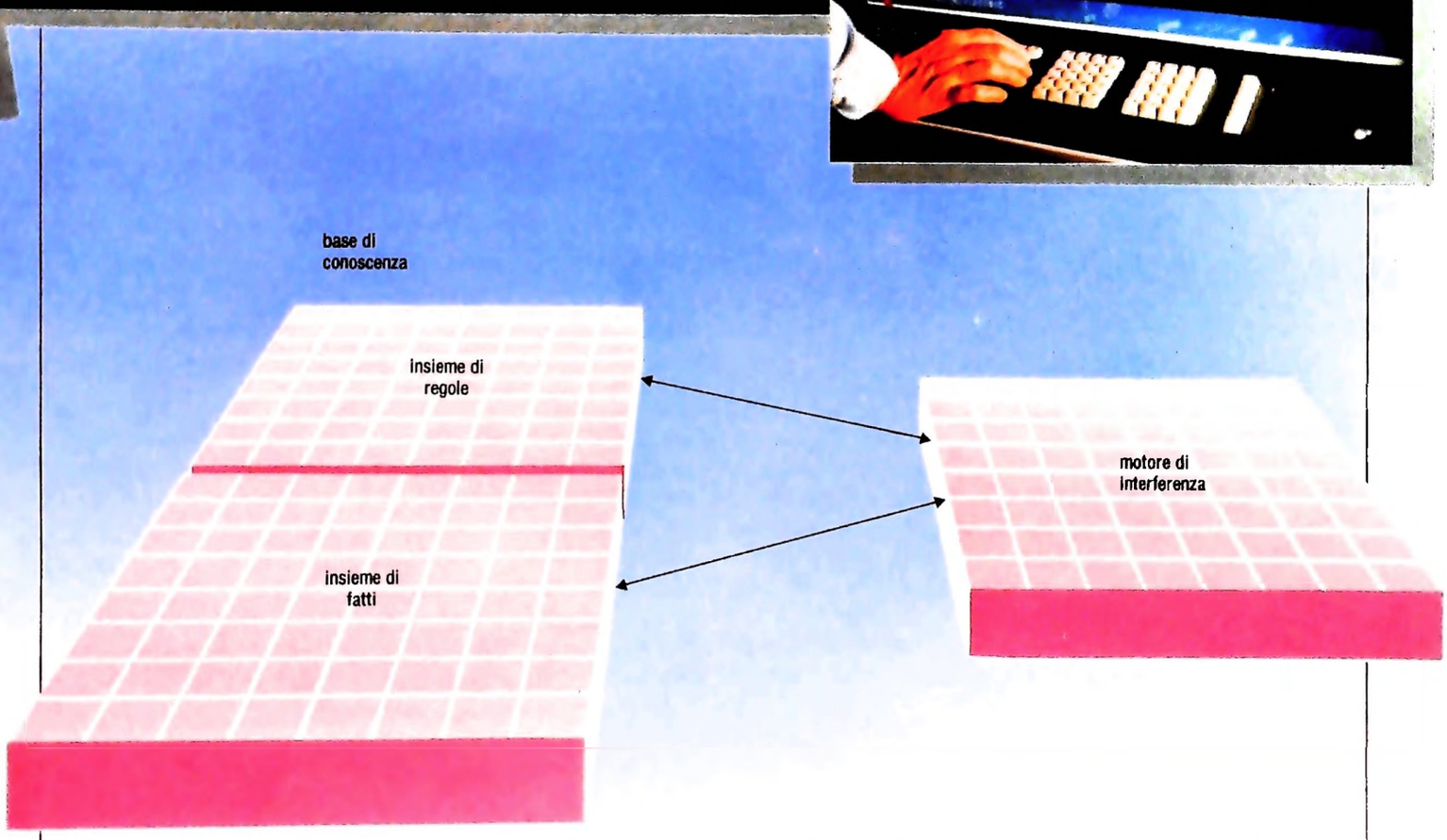
Ciò non significa che i sistemi esperti siano una panacea universale: in primo luogo per molti problemi non è necessaria alcuna capacità di ragionamento esperto, ma esiste un procedimento algoritmico fissato che conduce alla soluzione; in secondo luogo il costo di sviluppo di un sistema esperto può essere così alto, in termini di risorse umane e tecnologiche investite, da non essere sufficientemente motivato per la particolare applicazione.


## La struttura dei sistemi esperti

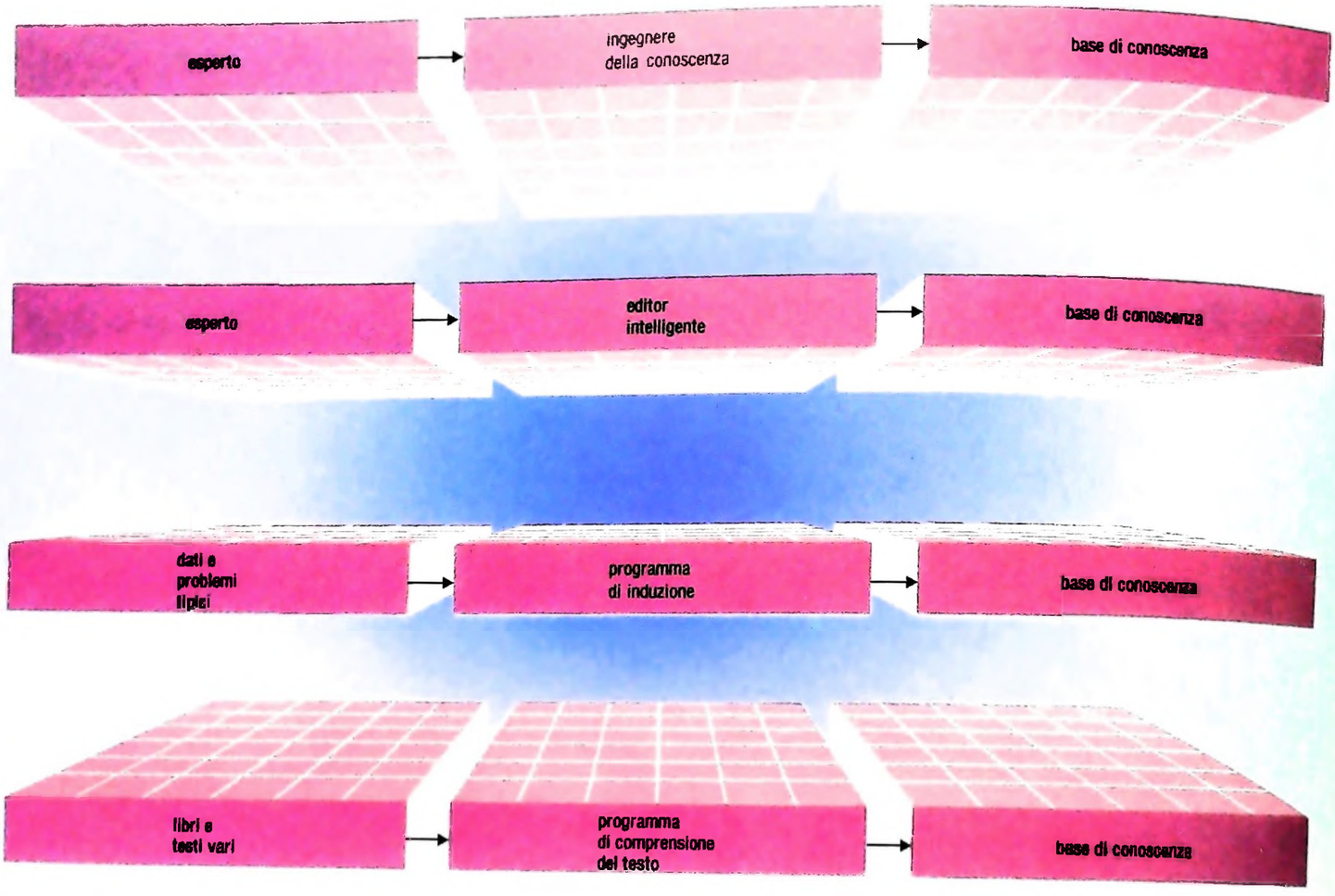
Da quanto detto emerge che un sistema esperto deve possedere una *base di conoscenza* specifica relativa al proprio campo di applicazione. Anche in domini molto ristretti le dimensioni della conoscenza di un esperto possono essere enormi: si pensi per esempio a un qualsiasi sottocampo della medicina. Questa base contiene la conoscenza relativa al problema affrontato: secondo un metodo di organizzazione che è stato spesso adottato nei sistemi sinora sviluppati (ma non è l'unico possibile) essa è costituita da un insieme di *fatti* noti sulla materia (per esempio tabelle, valori tipici di parametri significativi, mappe ...) e da un insieme di *regole*. Le regole codificano delle inferenze che possono essere fatte sul problema in esame o prescrivono particolari operazioni; si può dire che esse rappresentano quella parte del processo di ragionamento strettamente legato al compito svolto dal sistema:
"se il parametro B ha un valore maggiore di 400 e il parametro C ha un valore minore di 3500, allora è probabile al 75% che il paziente sia affetto da ...".
La base di conoscenza (nel seguito BdC) incorpora tutta la competenza specifica sul problema considerato. Perché queste nozioni possano essere utilizzate in un caso reale di consulenza esperta sono necessarie due cose:
1) che la BdC (regole e fatti) venga rappresentata fisicamente in modo adatto a essere manipolata e trasformata da parte del calcolatore;
2) che una parte del sistema sia dedicata a trasformare la BdC (a esempio applicando una regola e deducendo così nuovi fatti che possono dar luogo all'applicazione di altre regole), sino a risolvere il problema considerato, seguendo una strategia, o schema di ragionamento, generale. Secondo la terminologia corrente questa parte del sistema è detta *motore di inferenza* (schema della pagina accanto).
Il primo punto si presta a due interessanti osservazioni.
Intanto, si è parlato di rappresentazione della conoscenza in forma trattabile dal calcolatore. Poiché un sistema esperto è di fatto un complesso insieme di programmi scritti in qualche linguaggio (eventualmente un linguaggio appositamente disegnato), questo linguaggio deve possedere strutture dati e primitive adatti alla *manipolazione simbolica*. Per esempio può essere necessario accedere il conseguente di una regola (*Allora* ...) e sostituire, nel modo più efficiente possibile, alcuni acquisti. Esistono linguaggi che permettono più facilmente di altri questo genere di operazioni, in linea di principio fattibili anche con il BASIC, ma che di fatto risulterebbero troppo pesanti e poco efficienti: tra di essi ha assunto un ruolo primario il LISP, tanto che la totalità dei sistemi esperti sinora prodotti sono scritti in questo linguaggio o in qualche suo derivato. Tre dei prossimi articoli saranno dedicati a questo insieme di linguaggi e alle macchine che possono utilizzarli efficientemente.
Il secondo fatto importante è che la BdC del sistema esperto deve essere creata dal nulla sino ad arrivare a un accumulo di conoscenza sufficiente per affrontare brillantemente i compiti di un esperto. La costruzione della BdC corrisponde a una fase di "apprendimento" per il sistema. Questo processo è probabilmente la fase più critica nello sviluppo di un sistema esperto: una conoscenza inesatta o una codifica errata possono infatti degradare il rendimento complessivo ben al di sotto di prestazioni esperte. La strada attualmente seguita nell'acquisizione della conoscenza è questa: un tecnico del campo (detto *ingegnere della conoscenza*) interagisce con uno o più esperti della materia oggetto di studio. Durante questi incontri il tecnico sottopone all'esperto problemi tipici e lo segue nel processo di risoluzione. Una stretta interazione permette (dopo lunghi sforzi) di stabilire dapprima un lessico tecnico preciso sul problema e successivamente una schematizzazione dei metodi e dei procedimenti utilizzati dall'esperto per affrontare i problemi. In generale l'esperto utiliz-



GENERAL ELECTRIC




Per la creazione di un sistema esperto è indispensabile una base di conoscenza traducibile nel linguaggio del computer, che raccoglie le regole e i fatti; un motore di inferenza potrà poi inferire, su questa base, fornendo risposte a domande specifiche, che normalmente richiederebbero la consulenza e il parere di un esperto. Lo sviluppo di questa tecnica potrà modificare profondamente non solo il campo della ricerca scientifica, ma anche quelli della gestione, della previsione e dell'analisi nel settore commerciale. Nelle foto, esempi di utilizzazioni del computer nella ricerca e nella diagnosi medica.




La difficoltà maggiore, che si riscontra nei sistemi esperti consiste nel creare la base di conoscenza, cioè tutto quell'insieme di dati che permette al computer di affrontare i problemi che gli sono sottoposti. Nello schema sono rappresentate tre possibili strade per arrivare a questa acquisizione.

zerà sia regole *euristiche*, ovvero dettate dalla propria esperienza e intuito, che fanno parte della conoscenza specifica al problema, sia capacità strategiche generali di risoluzione. È compito dell'ingegnere della conoscenza enucleare queste informazioni e incorporarle secondo la corretta rappresentazione nella BdC o nel motore di inferenza del sistema esperto. Naturalmente la strada descritta non è così piana, ma comporta diversi tentativi e revisioni.

Si può quindi immaginare quanto sia pesante la fase di acquisizione della conoscenza e perché essa richieda talvolta anche anni di lavoro altamente qualificato. È naturale perciò che si cerchino modi alternativi o ausiliari nel ciclo descritto. Citiamo tre possibilità, anche se nessuna di esse è ancora pienamente utilizzabile (schema in alto):

1) si può pensare di alleviare il compito dell'ingegnere della conoscenza facendo sì che l'esperto conversi direttamente con un programma che codifica la conoscenza (si tratterebbe sostanzialmente di un editor molto sofisticato). Per fare questo è necessario un alto livello di comunicazione, possibilmente in linguaggio pseudo-naturale;

2) si possono indurre nuovi fatti e regole, associazioni causali automaticamente da un insieme sparso di dati, utilizzando osservazioni statistiche e casi tipici di problemi generati dal sistema stesso. Questo eviterebbe la complessa estrapolazione manuale di nuove regole;

3) l'ipotesi più futuribile è che un programma di comprensione di testi costruisca direttamente parte della base di conoscenza. Una parte di conoscenza legata all'esperienza non sarebbe catturata in questo processo, ma i vantaggi sarebbero comunque notevolissimi.

Un sistema esperto, ancora più di altri programmi, è soggetto a un'evoluzione vitale. Mano a mano che le competenze in un particolare campo crescono esso deve adeguarsi per mantenere i propri livelli di prestazione. È quindi fondamentale che la base di conoscenza del sistema possa venire agevolmente modificata quando se ne presenta la necessità: ciò può avvenire attraverso l'ingegnere della conoscenza, che introduce esplicitamente nuove regole o, come è sperabile in futuro, il sistema stesso può avere un "apprendimento" proprio, estendendo automaticamente le proprie competenze mano a mano che vengono risolti problemi specifici: anche in questo campo si può dire che l'A.I. si trova ai primordi.

*Lezione 30*

## I file in Basic

Vedremo in questa lezione un semplice esempio di uso di file in BASIC.
A tale scopo costruiremo un programma che memorizza nominativi su un file e successivamente li visualizza sul display.
Osserviamo lo schema del programma:

```
Apre file per registrazione;
Chiede nominativo;
WHILE NOT "stop" DO
        BEGIN
                registra nominativo su file;
                chiede nominativo
        END;
Chiude file;
Apre file per lettura;
Legge file
IF NOT EOF (file) THEN
        REPEAT
                Visualizza nominativo
                Legge file
        UNTIL EOF (file)
Chiude file
```

Esso mostra alcune novità ancora non esaminate.
Il file usato per le registrazioni viene "aperto" in due momenti del programma:
- una prima volta specificando che il file verrà usato per effettuare registrazioni;
- una seconda volta indicando che verrà usato per leggere le informazioni in esso memorizzate.

Non è possibile infatti effettuare operazioni di lettura e di registrazione contemporaneamente, bensì si deve operare all'interno di "sessioni di lavoro" nell'ambito delle quali è ammessa una sola delle due possibili operazioni, che viene dichiarata all'inizio della sessione stessa.
Per cambiare tipo di operazioni è necessario chiudere la sessione in corso e riaprirne una nuova: ciò ha anche l'effetto di "riavvolgere" il file, ovvero di riposizionarci sul primo elemento.
Così la CLOSE effettuata alla fine delle operazioni di registrazione permette di terminare la sessione attuale, ponendo il file in condizione di essere nuovamente trattato in una successiva sessione che lo vedrà automaticamente dall'inizio e non dalla posizione in cui abbiamo registrato l'ultimo elemento.
Alcune osservazioni vanno spese per quanto riguarda la struttura di programma usata per la gestione di fine file.
A questo proposito è bene innanzitutto osservare che i controlli di EOF vanno comunque effettuati dopo ogni operazione di lettura e in particolare anche dopo la lettura del primo elemento del file: esso infatti potrebbe essere "vuoto", cioè privo di qualsiasi elemento.
Se esaminiamo però la struttura di programma usata, converremo facilmente sul fatto che altre strutture anche più semplici potevano essere impiegate, come per

esempio la seguente:

```
Legge file;
WHILE not EOF (file) DO
        BEGIN
                Visualizza nominativo;
                Legge file
        END;
```

che ha il vantaggio di evitare la struttura IF esterna al ciclo di lettura ed è quindi innegabilmente più semplice e anche più efficiente poiché usa un numero minore di istruzioni. Perché dunque complicarsi tanto la vita?

Immaginiamo per un momento di voler avvertire l'utente del programma che il file è vuoto e che non sono quindi disponibili nominativi: dovremo poter distinguere questo caso da quello in cui, dopo la lettura di un certo numero di elementi, si arriva alla fine del file.

Nel primo caso infatti visualizzeremo un messaggio appropriato e abbandoneremo l'elaborazione, che verrà invece effettuata solo nel secondo caso.

La struttura del programma sarebbe allora la seguente:

```
Legge file;
IF NOT EOF (file) THEN
        REPEAT
                Visualizza nominativo;
                Legge file
        UNTIL EOF (file)
ELSE
                Visualizza messaggio;
```

Come si vede il criterio di scelta tra strutture di programma che effettuano le medesime operazioni può essere delicato. Nel nostro caso si trattava di scegliere tra una struttura più semplice e una più modificabile: quest'ultimo aspetto è sicuramente da tenere sempre in considerazione poiché sono frequenti, nella "vita" di un programma, i momenti in cui esso deve essere modificato per rimanere adeguato al variare dell'ambiente in cui si colloca.

Come si vede non abbiamo fatto altro che completare la prima struttura con la clausola ELSE.

Ma come avremmo potuto risolvere il medesimo problema se avessimo invece adottato la seconda soluzione?

Saremmo stati costretti a una scelta comunque infelice:

- inserire una variabile booleana che indichi se la lettura effettuata è la prima e in tal caso emettere il messaggio in caso di fine file;
- distruggere la struttura esistente e sostituirla con l'altra.

## Il programma Basic: la registrazione dei dati

Partiamo dalla realizzazione della parte di programma che effettua le registrazioni:

```
5 ' Inserisce nominativi nel file
10 N$="alfa"
15 OPEN"RAM:NOMIN" FOR OUTPUT AS #1
20 OPEN "RAM:N$" FOR OUTPUT AS #1
25 ' While not "stop" do
30 IF A$="stop" GOTO 100
40 PRINT #1,A$
50 INPUT "nominativo";A$
60 GOTO 30
100 ' Endwhile
110 CLOSE #1
```

(Si noti che il carattere ♯ è ottenuto premendo il tasto £)
e facciamo alcune osservazioni.
*L'apertura del file.*
È effettuata dall'istruzione OPEN che specifica:
• il nome del file, racchiuso tra doppi apici.
Il nome è costituito da un prefisso che specifica la periferica su cui cercare il file: vedremo nel seguito il significato dei vari prefissi, ma per il momento ci limiteremo a distinguere tra file residenti in RAM e file residenti su cassetta.
Il prefisso è seguito da i due punti e dall'identificatore del file;
• l'indicazione della modalità di accesso al file, introdotta dalla parola chiave FOR; specificando l'opzione OUTPUT dichiariamo di voler trattare il file in registrazione, con INPUT dichiariamo di volerlo elaborare in lettura;
• un identificatore numerico del file, che ricompare in tutte le successive istruzioni di accesso al file stesso. Tale identificatore individua univocamente il file all'interno di una sessione di lavoro.
*L'acquisizione di dati.*
Come al solito per fornire dati al programma da tastiera abbiamo usato l'istruzione INPUT. Poiché l'intenzione è di registrare nominativi abbiamo indicato una variabile di tipo stringa.
Naturalmente il programma non può sapere quanti nominativi potremo inserire: abbiamo perciò bisogno di una stringa speciale che il programma riconoscerà come indicatore di fine inserimento. Nel nostro caso è stata scelta la stringa "stop", che l'utente del programma dovrà digitare quando tutti i nominativi sono stati inseriti per bloccare la richiesta di dati dal programma e consentirne il proseguimento.
*La registrazione dei dati.*
I nominativi che di volta in volta forniamo al programma vengono quindi registrati nel file con l'istruzione PRINT della riga 40.
L'istruzione PRINT è ormai una vecchia conoscenza, ma in questo caso è stata usata in un formato nuovo, che specifica il numero di riconoscimento del file. L'effetto di questa informazione devia l'attenzione della PRINT dal display e la dirige verso l'area di memoria contrassegnata dal numero indicato.
Ad ogni successiva PRINT viene aggiunto un nuovo elemento al file, che è costituito di stringhe di caratteri.
*La chiusura del file.*
Al termine delle operazioni di registrazione la CLOSE dichiara la conclusione della attuale sessione di lavoro aprendo così la possibilità di aprirne una nuova.

L'esecuzione della OPEN in OUTPUT ha l'effetto di rendere disponibile il file richiesto, che a questo punto comparirà nel menù principale con il suffisso DO

L'obiettivo di indicare due identificatori per uno stesso oggetto, di cui uno numerico, è quello di rendere il programma parametrico rispetto al file su cui viene eseguito. Supponiamo infatti di volere eseguire una stampa fatture: è possibile che i nostri archivi siano organizzati in modo che esista un file per ogni anno: in tal caso lo stesso programma di stampa può essere eseguito su l'uno o l'altro dei file disponibili: a questo punto l'identificatore numerico ha una funzione analoga a quella di una variabile, a cui viene assegnato un valore tramite la OPEN.

Attenzione! L'indicatore di fine inserimento non fa parte evidentemente dei nominativi da registrare e non la troveremo quindi sul file: è importante perciò sceglierlo accuratamente, poiché nell'ambito del programma viene trattato come una parola riservata, che ha un significato preciso per l'esecuzione e non fa quindi parte dei dati soggetti a elaborazione.

## Il programma Basic: la lettura del file

A questo punto possiamo passare alla costruzione della parte di programma che legge i dati registrati e li visualizza sul display:

```
120 ' Visualizza dati contenuti nel file
130 CLS
135 OPEN "RAM:NOMIN" FOR INPUT AS #1
137 ' If not eof then
140 IF EOF(1) GOTO 200
145 ' Repeat
```

```
150 INPUT #1,A$
160 PRINT A$
170 IF NOT EOF(1) GOTO 150
175 ' Until eof
180 PRINT "Fine dati"
185 ' Else
190 GOTO 210
200 PRINT "File vuoto"
210 CLOSE #1
220 KILL"NOMIN.DO"
230 END
```

Facciamo anche in questo caso alcune osservazioni.

*L'apertura del file.*

Come nel caso della registrazione il file è stato "aperto", con le stesse modalità ma specificando che la sessione di lavoro che si inizia è di lettura (INPUT).

*L'individuazione di fine file.*

Osserviamo adesso come sono stati realizzati i controlli di fine file (EOF): notiamo subito che, a differenza di quanto fatto nello schema del programma, il controllo viene effettuato prima di ogni operazione di lettura (INPUT).

La funzione EOF messa a disposizione infatti ci dice se è o no disponibile il prossimo elemento da leggere.

Si noti che la funzione EOF compare nelle istruzioni IF come espressione booleana a cui è cioè possibile associare il valore VERO o FALSO.

*La lettura dei dati.*

L'acquisizione dei dati è fatta ancora una volta dalla istruzione INPUT, che già tante altre volte abbiamo usato per l'inserimento di dati da tastiera. Come nel caso della PRINT, però abbiamo in questo caso specificato l'identificatore numerico del file, indicando così che l'origine delle informazioni deve essere cercata nella corrispondente area di memoria.

*La distruzione del file.*

Alla fine del programma il file generato mediante la OPEN viene distrutto con il consueto comando KILL, che qui è stato usato come una istruzione BASIC. Se torniamo quindi al menù principale dopo l'esecuzione del programma non vedremo comparire il file NOMIN.

## Cosa abbiamo imparato

In questa lezione abbiamo visto:

- Strutture alternative di programma per realizzare cicli di lettura su file sequenziali;
- la realizzazione di files in RAM nel BASIC di M10;
- le istruzioni BASIC per il trattamento dei file;
- OPEN in registrazione e in lettura;
- PRINT e INPUT rispettivamente per registrare e leggere dati su e da file;
- CLOSE per "terminare" un ciclo di operazioni su file;
- l'uso dell'istruzione KILL all'interno di un programma.

# LA COMPLESSITÀ DEI PROBLEMI

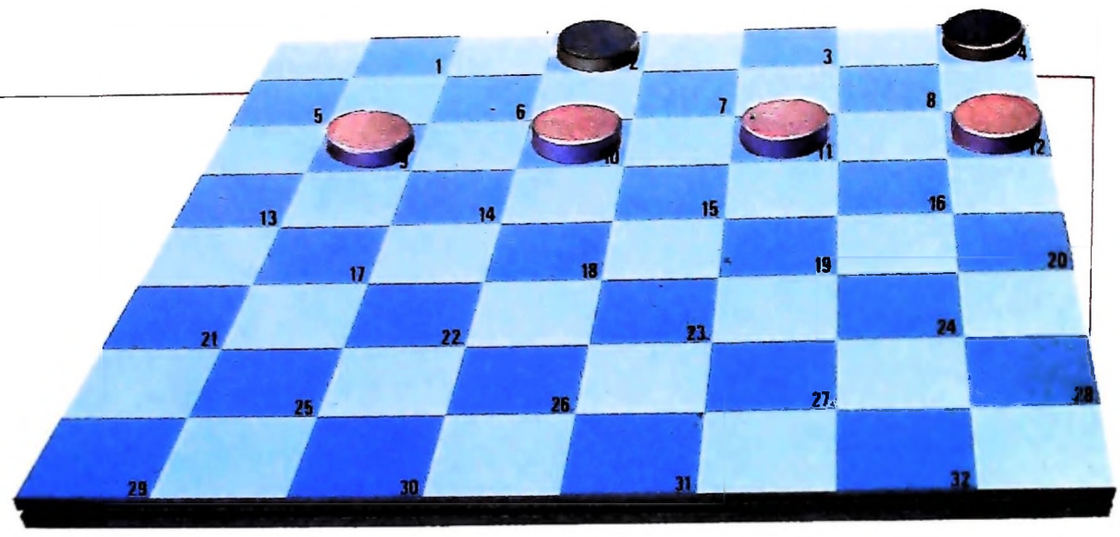
## Un'analisi profonda di ciò che gli elaboratori possono "realmente" fare.

### L'elaboratore e gli scacchi

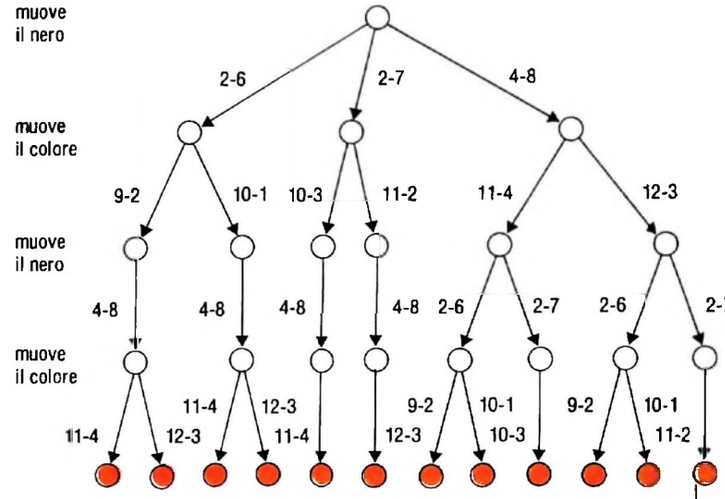
Esistono in commercio numerosi programmi che giocano a scacchi, ma nessuno di essi è in grado di battere un giocatore del livello di Karpov, anche se molti riescono a vincere contro giocatori di livello medio-alto. Eppure il problema di trovare una strategia di vittoria (ammesso che esista, e in tal caso esisterebbe per il bianco, che inizia per primo a muovere) in un gioco come gli scacchi è sicuramente un problema risolubile, e anzi sembrerebbe essere particolarmente adatto per un calcolatore.

Infatti, si tratta di trovare una successione di mosse tale che, per ogni possibile mossa del nero, esista una continuazione che porti il bianco alla vittoria. Per trovarla, è sufficiente costruire un "albero di gioco" formato da un insieme di punti collegati da frecce. Ogni punto corrisponde ad una disposizione dei pezzi sulla scacchiera, con l'indicazione del giocatore a cui tocca la mossa, mentre le frecce portano da un punto (configurazione) a tutte le configurazioni ottenibili con mosse lecite a partire da quella iniziale, e così via fino a raggiungere le configurazioni finali (scacco matto o patta). A questo punto è sufficiente contrassegnare le posizioni finali vincenti

Il calcolatore costituisce un buon avversario nel gioco degli scacchi e, teoricamente, può anche vincere, ma dovrebbe immagazzinare una tale quantità di mosse che in pratica un bravo giocatore avrà sempre la meglio




Come si sviluppa, nella dama, un "albero di gioco", che permette di determinare se l'esito del gioco sarà favorevole al nero, al colore o si risolverà a pari. Il primo punto del diagramma riprodotto a lato indica la posizione della pedina sulla scacchiera. Tutte le possibili mosse della pedina sono rappresentate dalla freccia, che individua sia la posizione della successiva mossa sia la direzione verso cui può essere indirizzata la pedina. Nello schema, dato che i punti della fila inferiore (colore) corrispondono a posizioni in cui non compare il nero, risulta che la posizione di partenza qui analizzata è vincente per il colore. La costruzione di un albero di gioco però non costituisce un metodo assoluto per garantire la vittoria, in quanto non è in grado di analizzare tutte le posizioni di partenza delle pedine: le dimensioni dell'albero infatti aumentano in modo esponenziale rispetto al numero delle pedine.




per il bianco e poi, risalendo indietro nell'albero, contrassegnare le configurazioni che consentono sempre, per qualunque contromossa del nero, di raggiungere una configurazione vincente. Il bianco può ora arrivare alla vittoria facendo mosse che lo portino sempre in una configurazione contrassegnata.

Un pezzo di albero di gioco relativo alla dama, gioco meno complesso, è rappresentato nella figura a lato.

Perché allora i calcolatori non riescono a vincere a scacchi? Il fatto è che sapere che un problema può essere risolto è ben diverso dall'essere in grado di risolverlo effettivamente. Nel caso degli scacchi, il numero di configurazioni possibili è talmente alto che nemmeno un calcolatore potentissimo sarebbe in grado di esaminarle tutte in un tempo ragionevole. Infatti, il numero di configurazioni cresce in modo esponenziale col numero di mosse fatte, poiché ogni configurazione ne genera numerose altre, ed ognuna di queste si riproduce a sua volta e così via; e abbiamo visto che la crescita esponenziale non è controllabile dagli elaboratori. Del resto, anche l'incauto sovrano che all'inventore del gioco degli scacchi promise un chicco di grano per la prima casella, due per la seconda, e via raddoppiando fino alla sessantaquattresima, si scontrò dolorosamente con la realtà della crescita esponenziale: alla sessantaquattresima casella infatti i chicchi di grano sono $16.10^{18}$, pari a 16000 miliardi di tonnellate. Consideriamo adesso un altro gioco, il "nim" o gioco dei fiammiferi. In questo gioco, si ha (nella forma più semplice) una fila di fiammiferi, per esempio 100, da cui ogni giocatore a turno toglie un numero variabile da 1 a n prefissato (a esempio 5) di fiammiferi. Perde chi è costretto a prendere l'ultimo fiammifero. Anche per questo gioco una strategia di vittoria si può cercare costruendo l'albero di gioco, il che ancora una volta richiede un tempo esponenziale ed è quindi impraticabile. Esiste però un algoritmo molto più semplice che porta alla vittoria il primo giocatore. È sufficiente infatti che il primo giocatore alla prima mossa faccia in modo che resti un numero di fiammiferi pari a un multiplo di n + 1 più un ulteriore fiammifero.

Nelle mosse successive dovrà invece prendere un numero di fiammiferi che, sommato a quelli presi dall'altro giocatore, dia n + 1. Così, ogni volta che il turno ritorna al secondo giocatore il numero di fiammiferi è diminuito di n + 1, finché resta l'ultimo fiammifero che il secondo giocatore è costretto a prendere.

## Problemi facili e problemi difficili

I due esempi fatti ci introducono a un secondo aspetto della teoria della complessità: la determinazione della complessità intrinseca dei problemi.

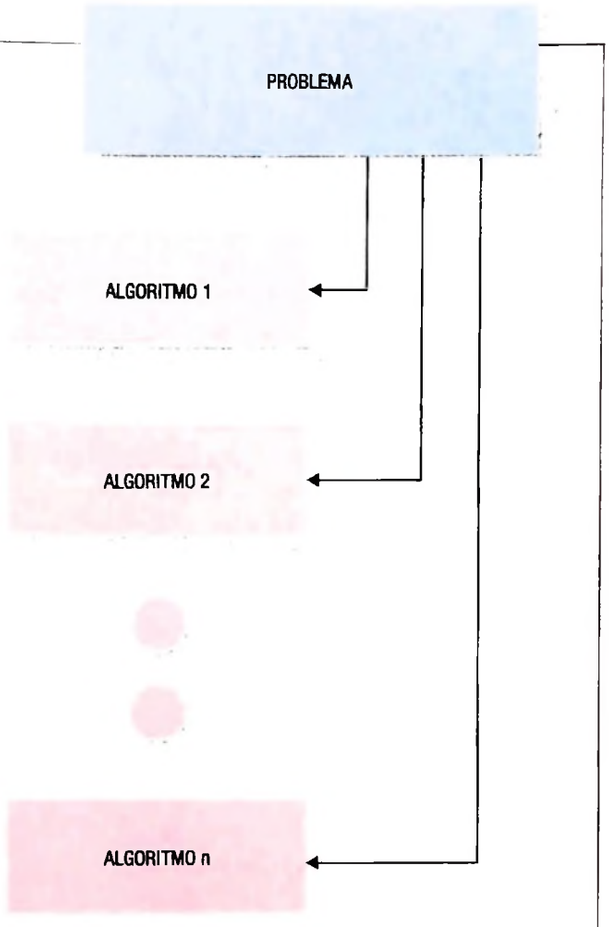
Come si è visto, un problema può essere risolto in più modi,

utilizzando diversi algoritmi. Poiché sappiamo misurare la complessità di un singolo algoritmo, possiamo definire la complessità intrinseca di un problema come la complessità dell'algoritmo più efficiente che lo risolve. Analogamente a quanto fatto per gli algoritmi, e con le stesse motivazioni, potremo poi porre la

**CONVENZIONE**

**Un problema è "facile" o praticamente risolubile se esiste almeno un algoritmo efficiente (polinomiale) che lo risolve. Un problema è "difficile" o intrattabile se non esiste nessun algoritmo efficiente che lo risolve.**

Mentre la teoria della computabilità si proponeva quindi di stabilire l'esistenza di almeno un algoritmo, buono o cattivo che fosse, per risolvere un dato problema, la teoria della complessità cerca di stabilire l'esistenza di buoni algoritmi, che consentano di risolvere in pratica i problemi.
A. Meyer e L. Stockmeyer hanno dimostrato che esistono problemi risolubili in linea di principio, ma per risolvere i quali anche il calcolatore più potente non solo esistente, ma neppure immaginabile (tenendo conto che le dimensioni dell'universo — quindi il numero di componenti dell'elaboratore — e la velocità della luce — e quindi la velocità di trasmissione delle informazioni all'interno del nostro elaboratore — sono finite) impiegherebbe almeno venti miliardi di anni, cioè più dell'età presumibile dell'universo.



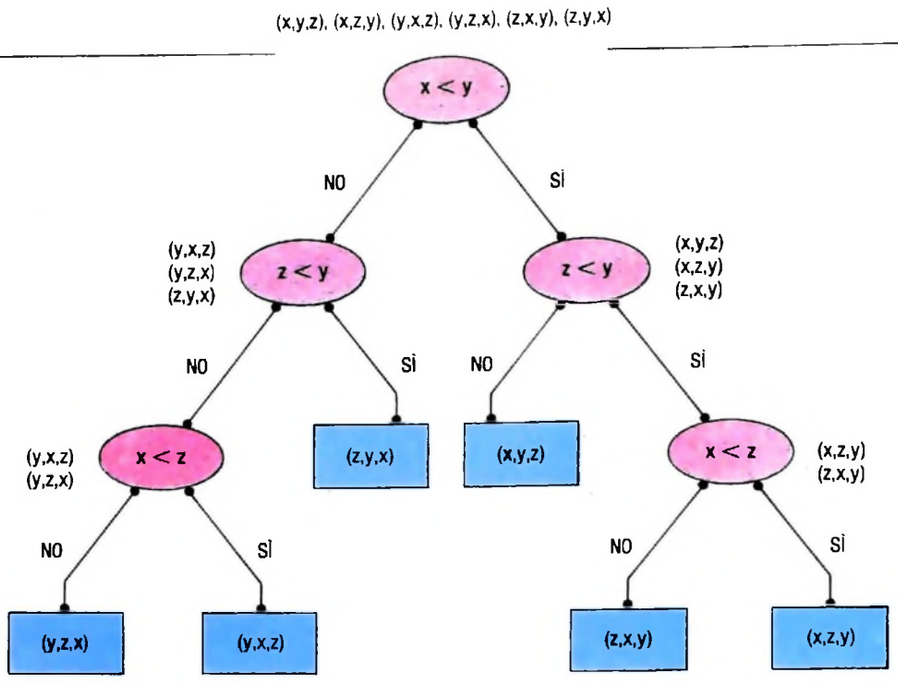

Un problema può essere risolto con più algoritmi diversi: è importante quindi cercare di stabilire quale sia il più conveniente, cioè quello che crea una minor quantità di passaggi.

## Le classi di complessità

In termini più formali, si possono raggruppare i problemi in classi di complessità e studiare quindi queste classi e le loro interrelazioni. Le principali classi studiate sono la classe P, contenente tutti i problemi che ammettono almeno un algoritmo polinomiale di soluzione, e la classe EXP dei problemi risolubili in tempo esponenziale, nonché la classe P-SPACE dei problemi risolubili in spazio polinomiale.
Un primo argomento di studio riguarda i rapporti di inclusione tra queste classi. Dal momento che un polinomio può essere visto come un caso particolare di esponenziale, si ha che $P \subseteq EXP$. D'altra parte, sono stati forniti esempi di problemi risolubili in tempo esponenziale, ma non in tempo polinomiale, per cui $P \neq EXP$.
Per la classe P-SPACE, si può osservare che un algoritmo utilizza un numero di celle di memoria non superiore al numero di mosse che esegue, per cui un algoritmo polinomiale in tempo è anche polinomiale in spazio: $P \subseteq$ P-SPACE. Analogamente, si può vedere che un numero polinomiale di celle offre la possibilità di eseguire al più un numero esponenziale di mosse, senza ritornare in una configurazione già assunta in precedenza (e quindi senza entrare in loop). Quindi P-SPACE $\subseteq$ EXP.
Tuttavia, non è ancora dimostrato che $P \neq$ P-SPACE. Anche se la cosa è poco verosimile, è possibile che ogni problema che ammette un algoritmo polinomiale in spazio ammetta anche un algoritmo polinomiale in tempo. Dare una risposta a questa domanda è uno dei principali problemi aperti della teoria della complessità.

## Limiti superiori ed inferiori di complessità

Veniamo adesso alla seconda questione: come si determina la complessità intrinseca di un problema P?
La prima cosa che possiamo fare è prendere tutti gli algoritmi di soluzione che conosciamo, valutarne la complessità e scegliere il più efficiente; supponiamo che questo abbia complessità $T_P(n)$. Ciò non significa però che abbiamo determinato la complessità intrinseca del problema: ne abbiamo semplicemente determinato un estremo superiore, cioè sappiamo che in ogni caso la complessità intrinseca non è superiore a $T_P(n)$. Potrebbe però essere inferiore; infatti, in genere non tutti i possibili algoritmi di soluzione di un problema sono noti: tra quelli che ancora non conosciamo potrebbe essercene uno di complessità inferiore a $T_P(n)$. Per esempio, nessuno è in grado di escludere che per gli scacchi sia possi-




A priori, l'ordinamento di tre numeri qualsiasi, *x*, *y*, *z* potrebbe avere sei esiti diversi, tutti ugualmente possibili. Per ordinarli si procede confrontandoli a due a due. Ogni confronto esclude un certo numero di possibilità. Quando gli esiti dei confronti effettuati hanno ridotto le possibilità a una sola, l'ordinamento è effettuato. Se gli elementi da ordinare sono *n*, per arrivare alla soluzione sono necessari, nel caso peggiore, $n.log_2 n$ confronti.

bile, come per il nim, trovare un algoritmo efficiente diverso dall'analisi brutale, e esponenziale, dell'albero di gioco.
Dobbiamo quindi trovare delle tecniche per dimostrare che una certa funzione T'P(n) è un estremo inferiore di complessità per il problema P, cioè che nessun algoritmo di risoluzione potrà mai scendere sotto tale complessità.
In alcuni casi, l'estremo inferiore di complessità può essere determinato in modo diretto, sfruttando alcune caratteristiche peculiari del problema P. È questo il caso del

### PROBLEMA DELL'ORDINAMENTO

Dati n numeri interi distinti, in un ordine qualsiasi, disporli in ordine crescente.

Per determinare l'estremo inferiore di complessità, ragioniamo nel caso in cui n=3. Partendo con 3 numeri x, y, z, nell'ordine, a seconda dei valori di questi numeri l'ordine in cui saranno dopo l'esecuzione dell'algoritmo è uno dei seguenti:

x, y, z ; x, z, y ; y, x, z ; y, z, x ; z, x, y ; z, y, x.

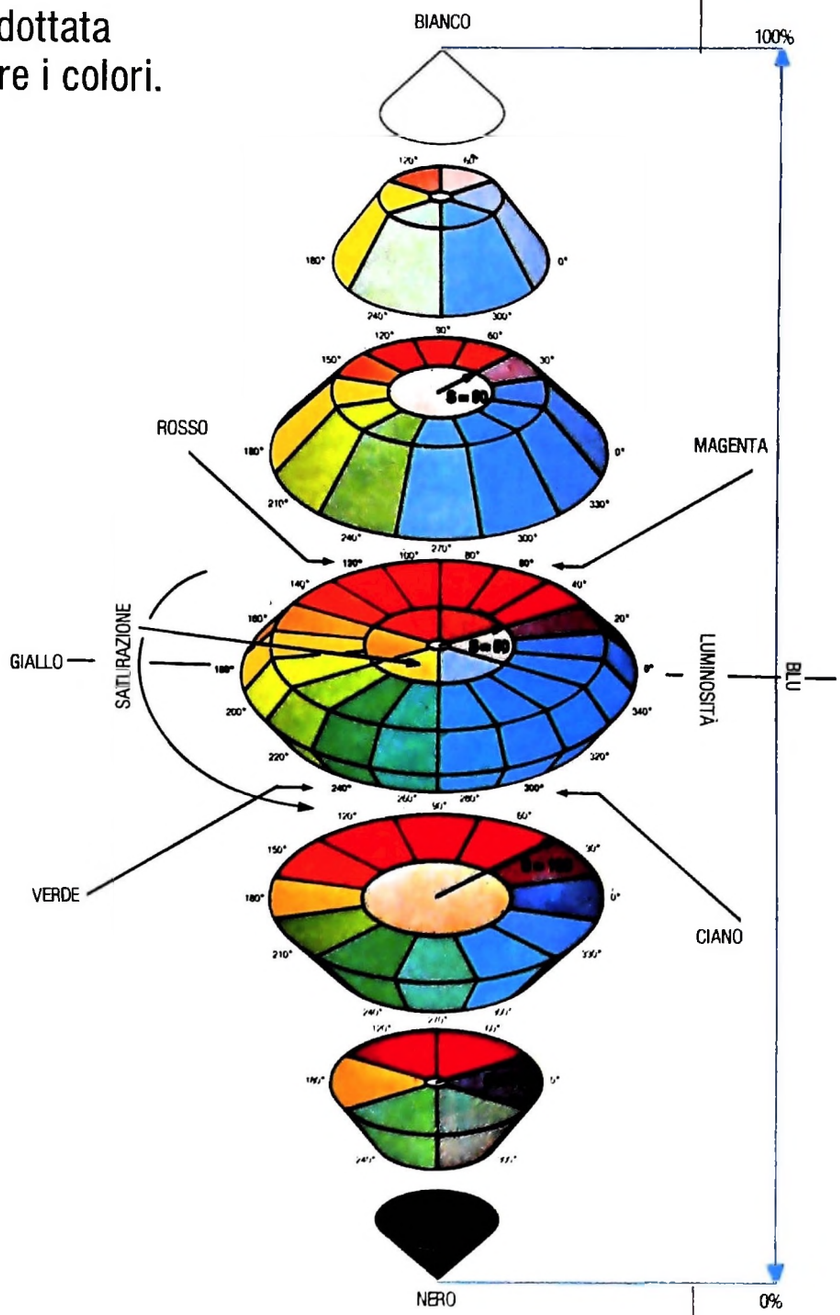
Per scegliere la successione corretta tra queste sei, dobbiamo procedere confrontando tra loro due dei numeri dati; l'esito del confronto consente di scartare un certo numero di possibilità.
Possiamo rappresentare tutte le possibili strade con un albero come quello della figura sopra. Ogni nodo corrisponde a un confronto, e quindi a una biforcazione. Poiché ogni confronto raddoppia (circa) il numero dei nodi, dopo k confronti si hanno $2^k$ nodi. Il problema è stabilire quanti confronti occorrono per avere almeno 6 nodi, tanti quante sono le possibilità iniziali, e si trova facilmente che ne occorrono 3.
Per n qualsiasi, si hanno n! = 1 · 2 · 3 ·... · (n-1) · n possibilità, e quindi occorrono almeno n · log n confronti per determinare univocamente l'ordine corretto. n log · n è quindi l'estremo inferiore di complessità per il problema dell'ordinamento.
Una tecnica diretta è stata utilizzata da M. Rabin, J. Hartmanis e R. Stearns per costruire esempi di problemi intrinsecamente esponenziali, utilizzando un meccanismo di diagonalizzazione analogo a quello ideato da Georg Cantor nel secolo scorso per dimostrare che i numeri reali non possono essere posti in corrispondenza biunivoca con i numeri interi.
In genere, sono però più utili le tecniche indirette. La principale tecnica indiretta utilizzata è la riduzione.
Diciamo che un problema P1 è riducibile a un problema P2 se esiste una funzione biunivoca f che trasforma ogni dato x di P1 in un dato y = f(x) di P2, con la seguente proprietà:
se la soluzione di P1 col dato di ingresso x è x' e la soluzione di P2 con ingresso y = f(x) è y', allora vale anche y' = f(x').
Ciò significa che per risolvere un problema P1 con un qualsiasi dato x possiamo prima applicare f a x, ottenendo l'elemento y = f(x), poi risolvere P2 con ingresso y, trovando y', e infine risalire, invertendo la funzione f, al valore x' tale che y' = f(x'), che è la soluzione di P1 con ingresso x.
Supponiamo adesso di sapere che P1 è un problema con un limite inferiore di complessità almeno esponenziale, e di dimostrare che P1 può essere ridotto a un secondo problema P2 attraverso una funzione f computabile in tempo polinomiale. In queste ipotesi possiamo concludere che anche per P2 non può esistere un algoritmo polinomiale di soluzione. In caso contrario infatti potremmo risolvere anche P1 in tempo polinomiale, contro l'ipotesi fatta, in questo modo: si riduce P1 e P2; si risolve P2; si trasforma il risultato nel risultato corrispondente di P1. Poiché i tre passaggi sono eseguibili in tempo polinomiale, l'intero procedimento è eseguibile in tempo polinomiale (la somma di tre polinomi è infatti un polinomio).

# I COLORI SUL VIDEO: IL FRAME BUFFER

## Approfondiamo la soluzione tecnologica adottata nei principali tipi di display per visualizzare i colori.

### Il tubo a raggi catodici dei display a colori

Nei display a colori il tubo a raggi catodici (CRT) è certamente più sofisticato rispetto a quello che è stato descritto per i monocromatici.
Ogni pixel dello schermo è costituito da una cella contenente tre tipi di fosfori disposti in modo triangolare, che, una volta eccitati, emettono colori diversi: uno rosso, uno verde e uno blu. Si ottengono, quindi, tre colori fondamentali, a partire dai quali, mediante opportune miscelazioni, si possono ottenere tutti i colori della scala cromatica.
Le singole celle, cioè i pixel, vengono chiamate "triadi" e sono così piccole che, quando le si guarda da una certa distanza, la luce emessa da un singolo fosforo si mescola con quelle degli altri due e l'osservatore percepisce la luce del colore risultante dalla fusione delle tre principali. A seconda quindi del grado di eccitazione cui viene sottoposto ogni fosforo di una triade, si possono ottenere una grande varietà di colori e le più lievi sfumature.
Ma come fanno i singoli fosfori ad essere eccitati dal fascio elettronico in modo differenziato?
Mentre nei display monocromatici esiste un unico cannone elettronico in grado di dirigere il fascio su un punto ben preciso dello schermo, nel caso dei display a colori i cannoni elettronici sono tre, disposti nello stesso modo triangolare delle triadi nei pixel. Inoltre fra la superficie dello schermo ed i cannoni elettronici è collocata una "maschera perforata", chiamata shadow-mask in inglese, che presenta un foro in corrispondenza di ogni pixel, ossia di ogni triade dello schermo.
I fori sono perfettamente allineati a una delle triadi, in modo tale che ognuno dei fosfori è esposto agli elettroni di uno solo dei tre fasci elettronici. Avviene quindi che l'insieme degli elettroni che costituiscono l'intero fascio generato dai tre cannoni controlla simultaneamente la luce rossa, verde e blu emessa dalla triade di fosfori che vengono eccitati dai singoli fasci (figura della pagina seguente).
La tecnologia del tubo a maschera perforata impone comunque, nonostante le dimensioni ridotte delle triadi, alcuni limiti nella risoluzione dell'immagine a colori che non è invece presente nel tipo monocromatico.

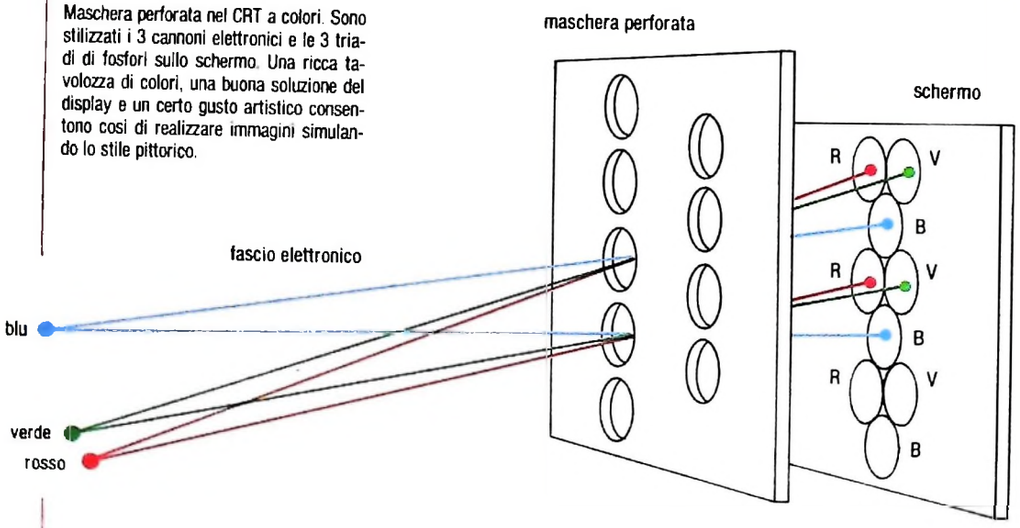



Standard del colore nella grafica, con i parametri di riferimento alle tre grandezze: tinta (i sei colori secondari), saturazione e luminosità. Il primo parametro varia da 0 a 360°, gli altri due variano da 1 a 100.

## Lo standard cromatico

L'eccitazione di un solo fosforo della triade dà origine a uno dei tre colori detti "principali"; l'eccitazione contemporanea di due genera rispettivamente altri tre colori: giallo = rosso + verde, ciano = verde + blu, magenta = rosso + blu, che sono chiamati "secondari"; l'eccitazione di tutti e tre dà origine alla luce bianca, mentre il nero si ottiene in assenza di eccitazione.
Così si ottengono gli otto colori fondamentali presenti in tutti i display a colori. Molto spesso però solo otto colori non sono sufficienti per soddisfare le esigenze applicative, per cui è necessario disporre di una tavolozza più ricca: a questo risultato si perviene mescolando fra loro, in percentuali diverse, i tre colori principali all'interno di ciascun pixel, dando così origine a sfumature diverse. Questo metodo consente di definire uno standard cromatico che sia univocamente identificabile sia dall'uomo che dalle macchine.
Per visualizzare questa situazione (figura della pagina precedente) dobbiamo pensare a un doppio cono in cui, fissato un qualsiasi strato circolare e muovendosi lungo la circonferenza esterna, si mescolano due colori adiacenti in percentuali diverse riuscendo così ad ottenere tonalità diverse; muovendosi lungo circonferenze più interne si mescolano tutti e tre i colori primari in percentuali diverse ottenendo dei colori sempre più sbiaditi, ossia meno "saturi", tendenti al grigio, che corrisponde al centro della circonferenza.
Sappiamo anche che ogni colore può essere più o meno brillante, ossia più o meno "luminoso", a seconda dell'intensità del fascio di elettroni incidente sui fosfori. Allora, variando la luminosità, si percorre il doppio cono dal basso all'alto e si ottiene un'ulteriore gamma di colori: dal nero, equivalente all'assenza di luminosità, sino al bianco, pari alla luminosità massima.

Maschera perforata nel CRT a colori. Sono stilizzati i 3 cannoni elettronici e le 3 triadi di fosfori sullo schermo. Una ricca tavolozza di colori, una buona soluzione del display e un certo gusto artistico consentono così di realizzare immagini simulando lo stile pittorico.




ACM-ARCHIVIO EIDOS

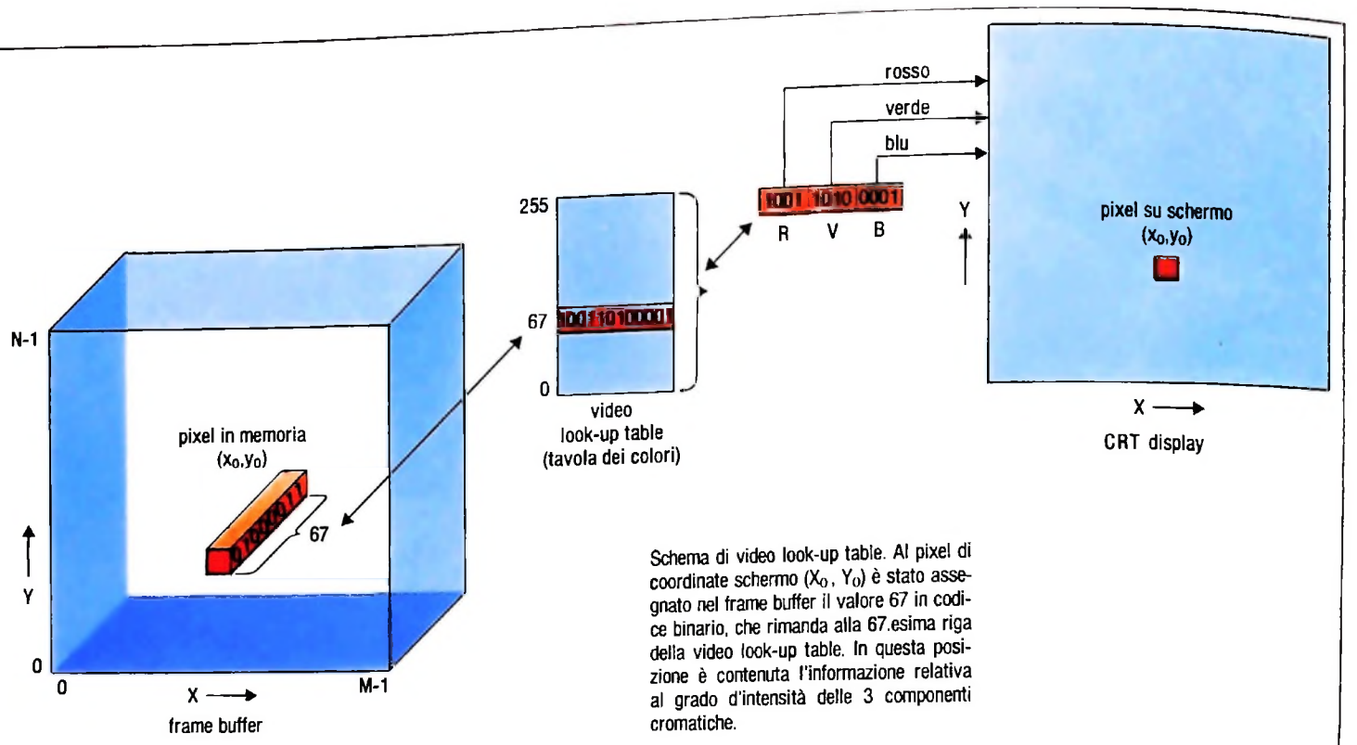
Questo standard cromatico, che è utilizzato nella computergrafica, viene detto HLS (Hue = tinta, Lightness = luminosità, Saturation = saturazione).
Una tavolozza può essere quindi definita in maniera univoca assegnando precisi parametri di riferimento alle tre grandezze che determinano i diversi colori. Tali grandezze e i relativi parametri sono:
*Tinta*: che varia da 0 a 360 gradi; questo parametro è usato per dare un nome a un colore (rosso, verde, azzurro ecc.) e ha quindi attinenza con la lunghezza d'onda della radiazione luminosa.
*Saturazione*: che varia dallo 0 al 100%; è il modo per indicare se un colore è puro. Si riferisce alla sua distanza rispetto al centro della circonferenza a cui appartiene; è anche detto "croma" o purezza.
*Luminosità*: che varia dallo 0 al 100%; è il modo per dire quando un colore è vivo o smorto e si riferisce alla sua posizione lungo l'asse verticale.
Ne consegue quindi che con il display a colori vi è la possibilità di rappresentare un numero praticamente illimitato di colori.

## Ancora sul display raster

Naturalmente la rappresentazione grafica a colori su un display è sotto controllo digitale e, di conseguenza, esistono limitazioni sul numero di colori che si possono generare. Inoltre, poiché informazioni visualizzate in colore diverso vengono raggruppate omogeneamente e memorizzate in un'area di memoria diversa, esistono anche limitazioni sul numero di colori utilizzabili contemporaneamente.
La potenzialità di un display grafico si giudica dal numero di colori disponibili sulla tavolozza, dal numero di colori utilizzabili contemporaneamente sullo schermo e dalla facilità con cui si possono modificare e definire i colori. I display a tecnologia raster, cioè a scansione lineare dei punti dello schermo, sono quelli che attualmente consentono di ottenere i risultati migliori.
La domanda che ci si deve porre ora è: come fa il computer a selezionare e a memorizzare i colori di un'immagine?
In un display raster l'immagine è generata pixel per pixel, assegnando a ciascuno di essi il particolare valore di intensità definito nella matrice in cui si è memorizzata l'immagine in forma digitale. Quindi in questa stessa matrice devono essere presenti le informazioni relative all'intensità e al colore di ogni pixel dell'immagine da visualizzare.
Un display a colori è costituito allora da tre componenti principali:
1) un'area di memoria digitale, in cui si memorizzano i valori delle intensità e i colori di ogni pixel, che viene chiamata "frame buffer";
2) un monitor televisivo sul quale visualizzare l'immagine;
3) un convertitore digitale/analogico la cui funzione è quella di scandire l'area del frame buffer e di produrre, in funzione dell'informazione attribuita a ogni pixel, un corrispondente segnale televisivo (ossia una certa intensità e direzione dei cannoni elettronici del CRT).

## Il frame buffer

Il frame buffer contiene quindi il valore dell'intensità di ogni pixel che forma l'immagine. Per esempio, per uno schermo con risoluzione 480 × 640 il numero totale di pixel è 307.200: il numero di bit dedicati a ogni pixel costituisce la misura delle possibili intensità che si possono assegnare. Se viene assegnato un solo bit, esso può assumere solo i due valori 0 e 1, per cui due gradi di intensità luminosa: pixel spento, pixel acceso. Se ne vengono assegnati 4, l'intensità può essere scelta fra $2^4 = 16$ livelli; il massimo numero di bit sinora utilizzati per ogni pixel in un frame buffer è 24, ed è usato per display di altissima qualità, molto costosi e capaci di generare una varietà di toni di colore dell'ordine di 16 milioni.




Schema di video look-up table. Al pixel di coordinate schermo ($X_0$, $Y_0$) è stato assegnato nel frame buffer il valore 67 in codice binario, che rimanda alla 67.esima riga della video look-up table. In questa posizione è contenuta l'informazione relativa al grado d'intensità delle 3 componenti cromatiche.

Attualmente comunque la maggior parte dei frame buffer sul mercato assegna da 1 a 8 bit per ogni pixel.
Per inserire la gestione del colore, oltre che dell'intensità luminosa dei pixel, si possono utilizzare vari metodi. Il più semplice è di definire le componenti dei tre colori fondamentali che deve avere ogni pixel. I bit che rappresentano il pixel possono essere divisi in tre gruppi, ciascuno dei quali indica l'intensità di uno dei tre colori principali: per esempio, in un frame buffer di 8 bit per pixel, sono normalmente dedicati 3 bit per il rosso, 3 per il verde e 2 per il blu.
Questa gestione dei colori ha però lo svantaggio di offrire una tavolozza abbastanza limitata. Un sistema più flessibile che consente una più ricca scelta cromatica è quello che implica l'utilizzo della cosiddetta "tavola dei colori" o "video look-up table".

## La video look-up table

I valori memorizzati nel frame buffer in questo caso vengono trattati come riferimenti a una tabella di colori definiti dalle loro componenti di rosso, verde, blu. Un frame buffer con 8 bit per pixel, consente di avere riferimenti a una tabella di 256 colori, dato che sono $2^8 = 256$ i possibili valori attribuibili a ogni pixel nel frame buffer.
Una cella di memoria della look-up table è costituita a sua volta da un certo numero di bit, che vengono suddivisi in tre gruppi: uno per il rosso, uno per il verde e uno per il blu. I valori numerici in codice binario che assumono questi gruppi di bit corrispondono all'intensità da attribuire a ciascun colore, e cioè all'intensità luminosa di ogni fosforo della triade di un pixel.
Nella figura in alto si evidenzia con un esempio il funzionamento della video look-up table: al pixel che ha coordinate schermo ($x_0$, $y_0$) è stato assegnato nel frame buffer il valore 67 in codice binario, che rimanda alla 67-esima riga della video look-up table. In questa posizione è contenuta l'informazione relativa al grado di intensità delle tre componenti cromatiche, alle quali vengono assegnati 4 bit ciascuna. Il convertitore digitale/analogico provvederà poi a trasformare le informazioni contenute nella look-up table per controllare i tre cannoni elettronici del tubo catodico.

## Colori disponibili e colori simultaneamente disponibili

Abbiamo visto che ogni riga della tavola dei colori è suddivisa in tre gruppi di bit. Il numero di bit dedicati a ogni colore dà la misura delle possibili sfumature che si possono ottenere di quel colore. Naturalmente più sono i bit della singola riga, più numerose saranno le sfumature disponibili.
Il numero di colori simultaneamente disponibili sullo schermo però è dato dal numero di righe della tavola dei colori, che, ripetiamo, dipende dal numero di bit dedicati a ogni pixel nel frame buffer. Per esempio, nel caso illustrato nella figura, i possibili colori che si possono ottenere nella look-up table sono $2^{12}$, che è un valore molto più alto delle 256 righe disponibili. Questo significa che, all'inizio di ogni attività, la look-up table viene caricata automaticamente con una combinazione di 256 colori definita a priori, che costituirà la tavolozza dei colori disponibili simultaneamente; se poi, durante l'esecuzione del lavoro c'è la necessità di avere altre gamme cromatiche, bisognerà specificare al computer di "caricare" nella look-up table altri colori in sostituzione di quelli che non serviranno più.

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.

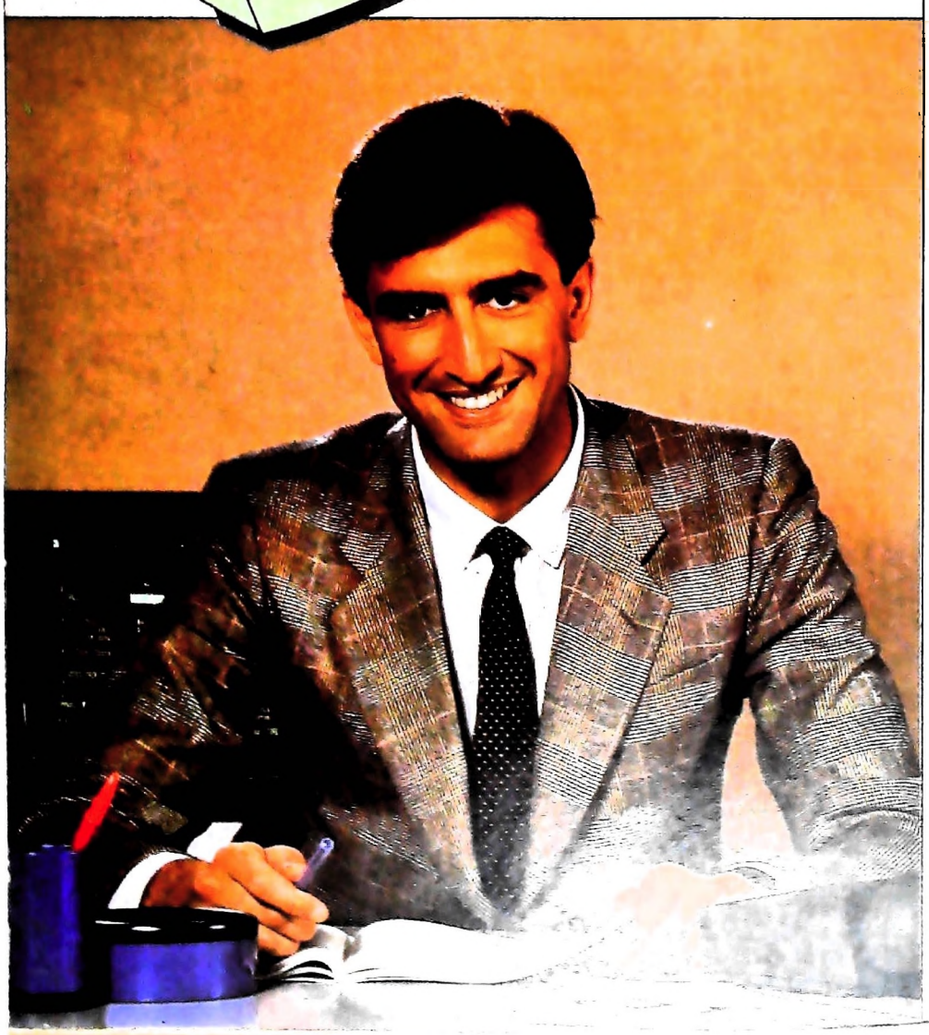
Basti pensare che tutte le unità base includono sia l'interfaccia seriale che quella parallela. Oppure basti pensare all'ampia gamma di supporti magnetici: floppy da 360 a 720 KB o un'unità hard disk (incorporata o esterna) da 10 MB.
La loro compatibilità, inoltre, fa sì che si possa far uso di una grande 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.



**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.