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

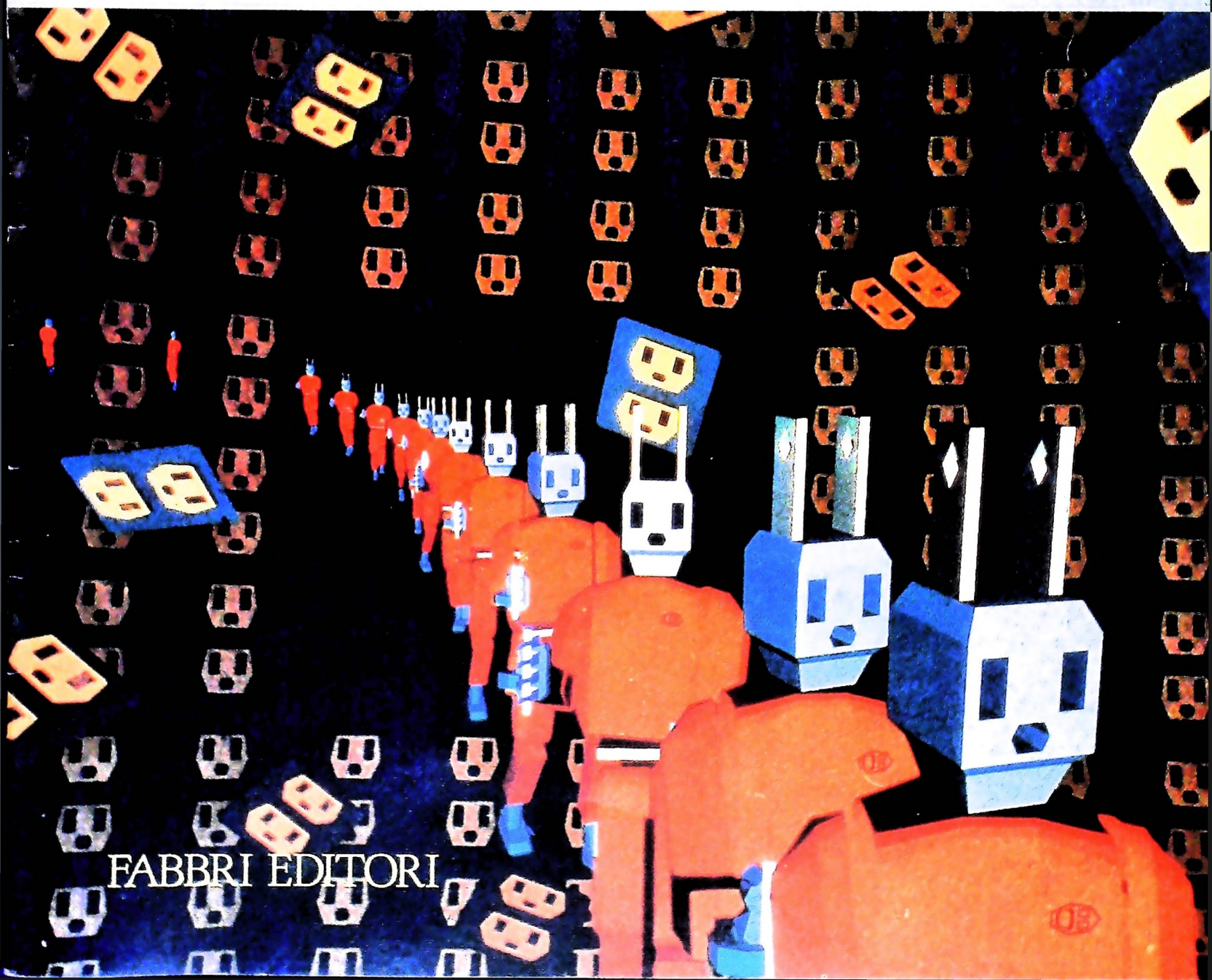
32 CORSO PRATICO COL COMPUTER

421842

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

F4 F5 F6 F7 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
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

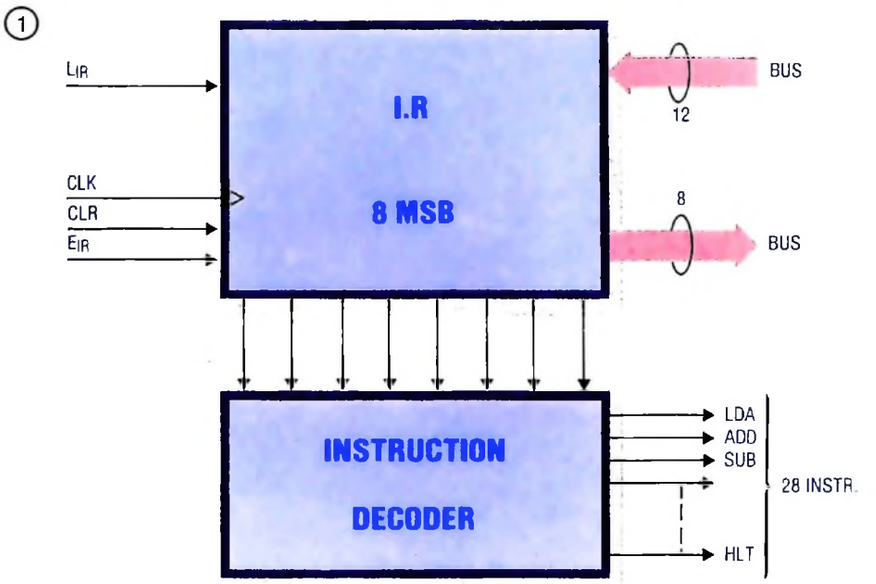
Impaginazione
BRUNO DE CHECCHI
PAOLA ROZZA

Programmazione Editoriale
ROSANNA ZERBARINI
GIOVANNA BREGGÈ

Segretarie di Redazione
RENATA FRIGOLI
LUCIA MONTANARI

I.R. (REGISTRO DI ISTRUZIONI)

Un componente del sistema di controllo la cui funzione è quella di decodificare le istruzioni.

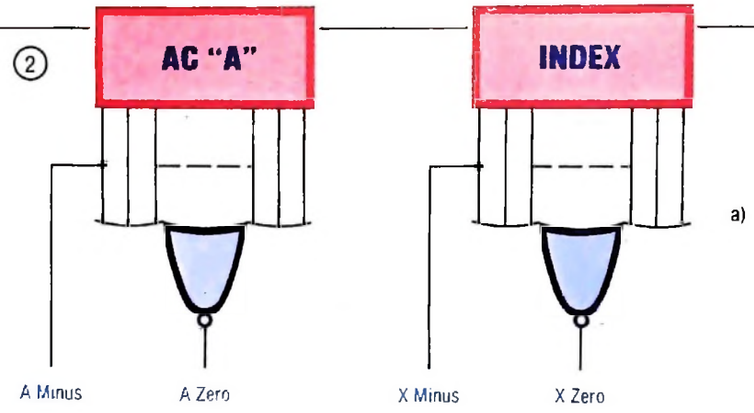


Il circuito completo di controllo formato da due blocchi: l'I.R (Registro Istruzioni) e l'INSTRUCTION DECODER (Decodificatore di Istruzioni).

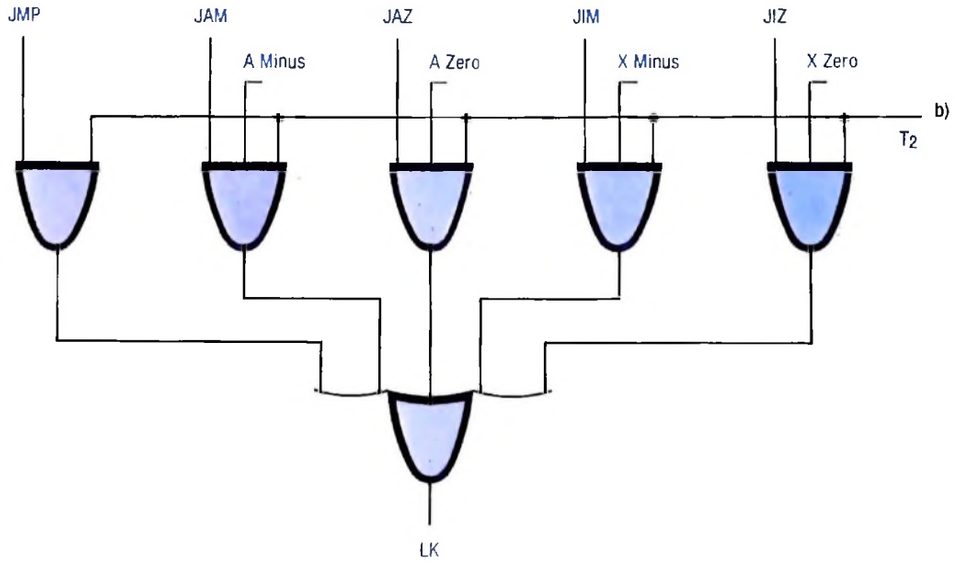
Nella figura 1 rappresentiamo il registro I.R che fa parte del sistema di controllo: vediamo com'è fatto.
La Uamicro II usa, come sappiamo, una parola di 12 bit, per cui nel registro I.R entrano 12 bit, naturalmente quando $L_{IR} = 1$.
Abbiamo visto nel modulo precedente che la parola poteva essere interpretata in tre modi:
— Istruzione più indirizzo.
— Istruzione "Operate".
— Dati.
La prima forma occupa i quattro bit più alti come istruzione e gli otto bit più bassi come indirizzo. La seconda usa i quattro bit più alti per indicare che si tratta di una istruzione "Operate" per cui i quattro bit più alti sono tutti 1 e i quattro che seguono servono per indicare il tipo di istruzione, mentre gli ultimi quattro bit non sono usati, per ora. La terza forma usa solo gli ultimi otto bit. Precisiamo comunque che i dati vengono considerati da sinistra verso destra; pertanto i bit più significativi sono quelli più a sinistra e i meno significativi quelli a destra.
Dopo questa premessa, possiamo capire più facilmente l'architettura dell'I.R. In esso entrano 12 bit: se l'istruzione è una MRI ne escono 8 come indirizzo, mentre i primi quattro vengono trasmessi al registro sottostante "INSTRUCTION DECODER" (decodificatore di istruzioni) che poi, a sua volta, attiverà la linea dell'istruzione corrispondente che entra direttamente nel CON. Se si tratta di un'istruzione di "Operate" l'intera parola di 12 bit resta internamente all'I.R per essere decodificata. Se infine si tratta di dati, questi entrano nei registri indicati.

Circuiti per istruzioni JMP

Le istruzioni di salto vengono realizzate quando una determinata condizione si avvera. La tecnica in generale consiste



Schemi di circuito completo per salti condizionali e incondizionali. A lato: circuiti che riconoscono il valore negativo o zero dell'Accumulatore e il Registro Index. Sotto: circuito che permette il verificarsi di salto condizionato quando sono date le condizioni giuste.



Qui sotto: i codici emessi dal controllo per la realizzazione delle istruzioni OPERATE. Nella pagina accanto, istruzioni della Uamicro II e le varie fasi.

3

	S_0	S_1	S_2	S_3	OPERATE
NOP	0	0	0	0	NO OPERATE
ADD	1	0	0	0	A ← (A + B)
SUB	0	1	0	0	A ← (A − B)
CMA	1	1	0	0	A ← $\bar{A}$
CMB	0	0	1	0	B ← $\bar{B}$
IOR	1	0	1	0	A ← (A "OR" B)
AND	0	1	1	0	A ← (A "AND" B)
NOR	1	1	1	0	A ← (A "NOR" B)
NAND	0	0	0	1	A ← (A "NAND" B)
XOR	1	0	0	1	A ← (A "XOR" B)
CLA	0	0	1	1	A ← 0

nell'uso di un registro, chiamato **STATUS**, i cui bit vengono modificati secondo determinati valori dell'accumulatore o di altri registri. Questa stessa tecnica è usata nella Uamicro II. Circostanze che possono provocare salti ce ne sono molte: alcune di loro, le più comuni, le vedremo nella Uamicro III; per ora ci limiteremo a dimostrare la tecnica hardware necessaria per poterla realizzare. Le condizioni della Uamicro II sono:

— AMinus = Accumulatore negativo.
— AZero = Accumulatore uguale a zero.
— XMinus = Registro INDEX negativo.
— XZero = Registro INDEX uguale a zero.

I circuiti usati per rivelare le suddette condizioni sono illustrati in figura 2a). Per quanto riguarda il valore uguale a zero dei due registri, questo è riconosciuto dai NOR, mentre il valore negativo è segnalato dal MSB = 1. Il quadro è completato dal circuito di figura 2b). Questo è un circuito combinatorio con tre entrate: da una parte ci sono le istruzioni, da

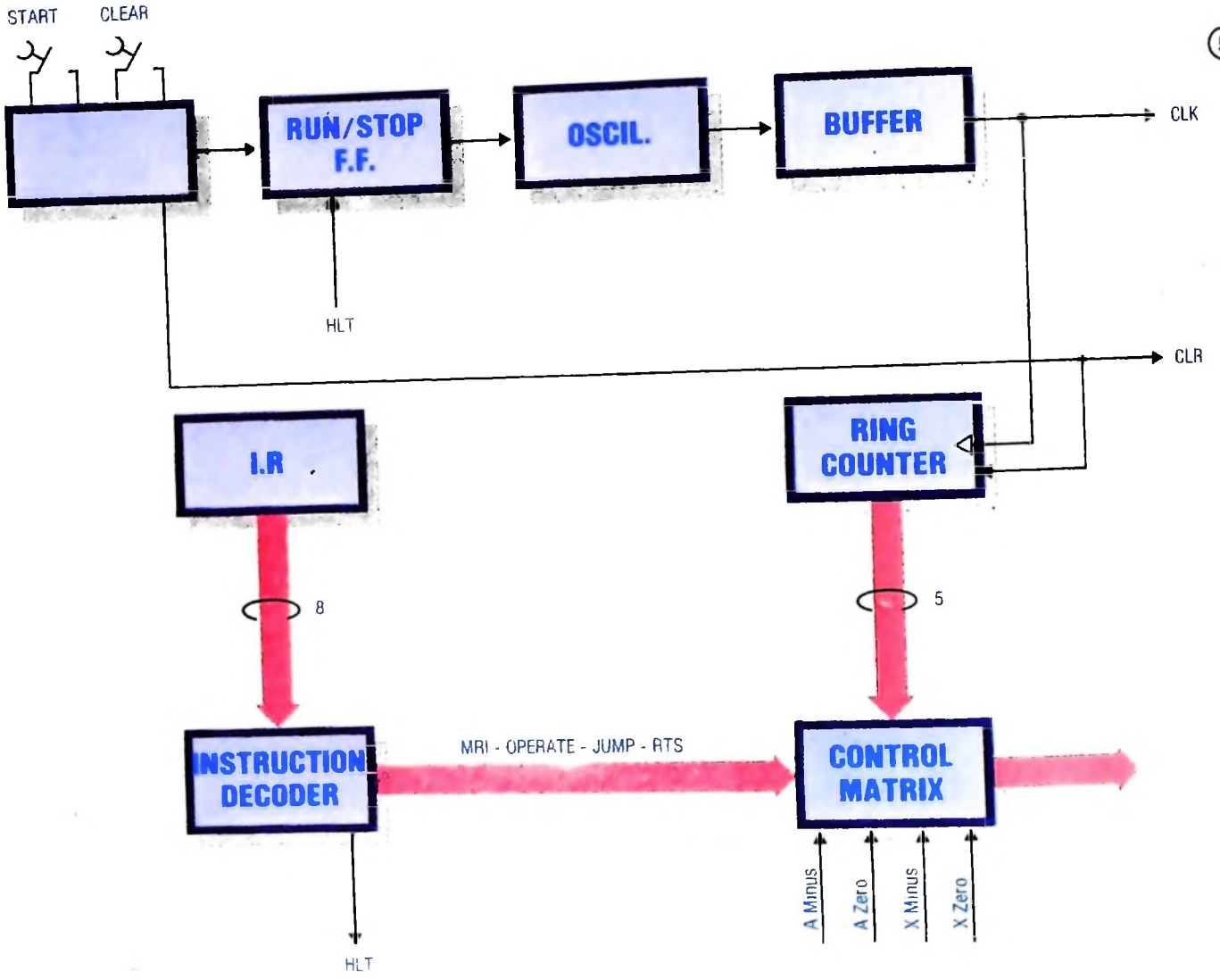
④

MNEM	FASE T_2	T_3	T_4	T_5	T_6
LDA	$E_{IR} \cdot L_M$	$CS \cdot L_A$			
ADD	$E_{IR} \cdot L_M / S_0$	$CS \cdot L_B$	$E_U \cdot L_A \cdot S_0$		
SUB	$E_{IR} \cdot L_M / S_1$	$CS \cdot L_B$	$E_U \cdot L_A \cdot S_1$		
STA	$E_{IR} \cdot L_M$	E_{AC}.W/R.CS			
LDB	$E_{IR} \cdot L_M$	$CS \cdot L_B$			
LDX	$E_{IR} \cdot L_M$	$CS \cdot L_X$			
JMP	$E_{IR} \cdot L_{PC}$ o L_{SC}				
JAM	$E_{IR} \cdot L_{PC}$ o L_{SC}	----------	--------	$A_{MINUS} = 1$	
JAZ	$E_{IR} \cdot L_{PC}$ o L_{SC}	----------	--------	$A_{ZERO} = 1$	
JIM	$E_{IR} \cdot L_{PC}$ o L_{SC}	----------	--------	$I_{MINUS} = 1$	
JIZ	$E_{IR} \cdot L_{PC}$ o L_{SC}	----------	--------	$I_{ZERO} = 1$	
JMS	P_U	$E_{IR} \cdot L_K$	--------		
NOP					
CLA	$E_U \cdot L_A \cdot S_3 \cdot S_2$				
XCH	$E_A \cdot L_{AR}$	$E_X \cdot L_A$	$E_{AR} \cdot L_X$		
DEX	DEX				
INX	INX				
CMA	$E_U \cdot L_A \cdot S_1 \cdot S_0$				
CMB	$E_U \cdot L_B \cdot S_2$				
IOR	$E_U \cdot L_A \cdot S_2 \cdot S_0$				
AND	$E_U \cdot L_A \cdot S_2 \cdot S_1$				
NOR	$E_U \cdot L_A \cdot S_2 \cdot S_1 \cdot S_0$				
NAN	$E_U \cdot L_A \cdot S_3$				
XOR	$E_U \cdot L_A \cdot S_3 \cdot S_0$				
RTS	P_D	----------	----------		
LXA	$E_{IR} \cdot L_{AC}$	$E_X \cdot L_B \cdot S_0$	$E_{AC} \cdot L_M$	$CS \cdot L_{AC}$	
SXA	$E_{AC} \cdot L_{AR}$	$E_{IR} \cdot L_A$	$E_X \cdot L_B \cdot S_0$	$E_{AC} \cdot L_M$	$E_{AR} \cdot W/R. CS$
INP	L_{IN}	$E_{IN} \cdot L_A$			
OUT	$E_A \cdot L_O$				

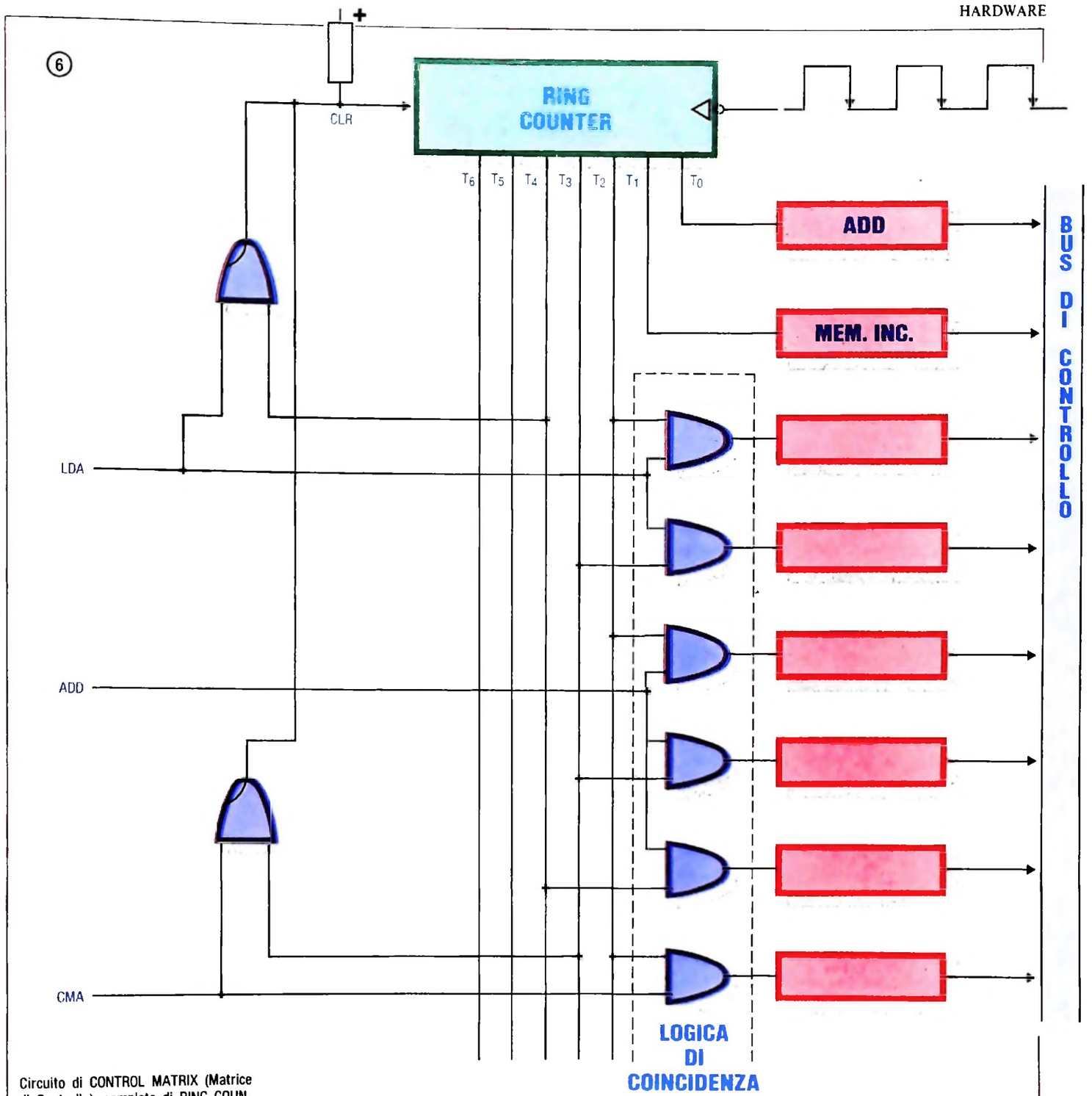
un'altra le condizioni e infine il tempo T_2, che è la fase in cui vengono autorizzate le condizioni per effettuare o meno i salti. L'uscita naturalmente è la L_K che entra nell'SC per attivare il circuito che detiene in quel momento il controllo, come abbiamo già visto.
Nella tavola di figura 3 presentiamo i codici per le varie operazioni, mentre nella tabella di figura 4, presentiamo le istruzioni con i vari comandi distribuiti nel tempo.

CON (Registro di Controllo)

Nella figura 5 presentiamo lo schema a blocchi del registro di controllo. Incominciando da sinistra verso destra c'è il primo blocco con i consueti contatti START e CLEAR, l'ultimo dei quali pone a zero i registri, mentre lo START attiva il secondo registro, cioè il RUN/STOP F.F.; questo, a sua volta, attiva l'oscillatore che continuerà a oscillare fin quando non riceve un segnale di HALT. Il quarto registro, il BUFFER, non è altro che un amplificatore di potenza. Il primo registro in basso a sinistra è l'I.R dove si fa la prima selezione, per cui le istruzioni di MRI - OPERATE- JUMP - RTS vanno al controllo, mentre l'istruzione HALT va direttamente al circuito flip-flop RUN/STOP per bloccare l'oscillatore.
Restano ora da analizzare gli ultimi due registri: il primo è il RING COUNTER, che già conosciamo bene, e il secondo il CONTROL MATRIX (matrice di controllo). Quest'ultimo non è uguale alla Matrice di Controllo della Uamicro I per-



Architettura del circuito di controllo completo.



Circuito di CONTROL MATRIX (Matrice di Controllo), completo di RING COUNTER (contatore ad anello) e di LOGICA DI COINCIDENZA (AND a due entrate).

ché, come avevamo detto, esistono altre forme per risolvere lo stesso problema, in special modo quando le condizioni incominciano a diventare numerose.
Una forma molto veloce ed efficiente è quella di creare parole di controllo fisse, visto che la parola di controllo per ogni fase, in corrispondenza dell'istruzione, non cambia, e attivarla nel momento giusto come è illustrato nella figura 6. Inoltre possiamo ridurre le fasi per ogni istruzione a quelle strettamente necessarie e infatti gli AND a collettori aperti ci permettono di ridurre le fasi azzerando il RING COUNTER (contatore ad anello). Da notare che le prime due fasi T_0 e T_1 sono quelle del FETCH, per cui attivano le due parole con le rispettive funzioni di AND (fase di indirizzi) e MEM. INC. (fase di memoria e incremento).

IL LINGUAGGIO PASCAL (I)

Un linguaggio estremamente facile da apprendere e molto spontaneo da usare.

Il linguaggio Pascal è opera di Niklaus Wirth, professore del Politecnico di Zurigo, che lo definì verso la fine degli anni Sessanta.
Esso può essere considerato un preciso punto di riferimento nella cultura dei linguaggi di programmazione: infatti raccoglie e sistematizza tutti i principali concetti sulla programmazione emersi nelle ricerche sul tema, e in particolare quelli della programmazione strutturata e dello sviluppo top down.
Un primo compilatore Pascal fu costruito dallo stesso Wirth nel 1970; successivamente il linguaggio si diffuse rapidamente su moltissimi calcolatori, in particolare su calcolatori destinati a un largo pubblico. Infatti, a dispetto del fatto che si tratta di un linguaggio piuttosto raffinato e ricco di concetti avanzati di informatica, risulta facile da apprendere e molto spontaneo da usare.
Il Pascal è attualmente uno dei linguaggi più diffusi, anche se spesso il suo uso per la costruzione di programmi in ambiente industriale è limitato (si conoscono tuttavia esempi di interi sistemi operativi sviluppati completamente in Pascal).

Caratteristiche generali

Il linguaggio Pascal è particolarmente orientato allo sviluppo top down dei programmi; la struttura di un programma Pascal è quella illustrata qui sotto.

- Una prima parte è destinata alla descrizione dei dati; questa comporta:
  - *definizioni*, cioè convenzioni linguistiche che facilitano al programmatore il compito di modellare le entità del problema; come vedremo meglio in seguito, è possibile definire nuovi tipi di dati oltre quelli già messi a disposizione dal calcolatore: per esempio, la definizione
    TYPE giorno = 1..31;
    individua un attributo che potrà essere associato a una variabile, indicando che questa potrà assumere solo valori compresi tra 1 e 31;
  - *dichiarazioni*, cioè richieste di predisposizione di opportune variabili, nel rispetto delle definizioni linguistiche precedenti; così è possibile dichiarare, come vedremo meglio in seguito, variabili di tipi normalmente disponibili in tutti i linguaggi, come

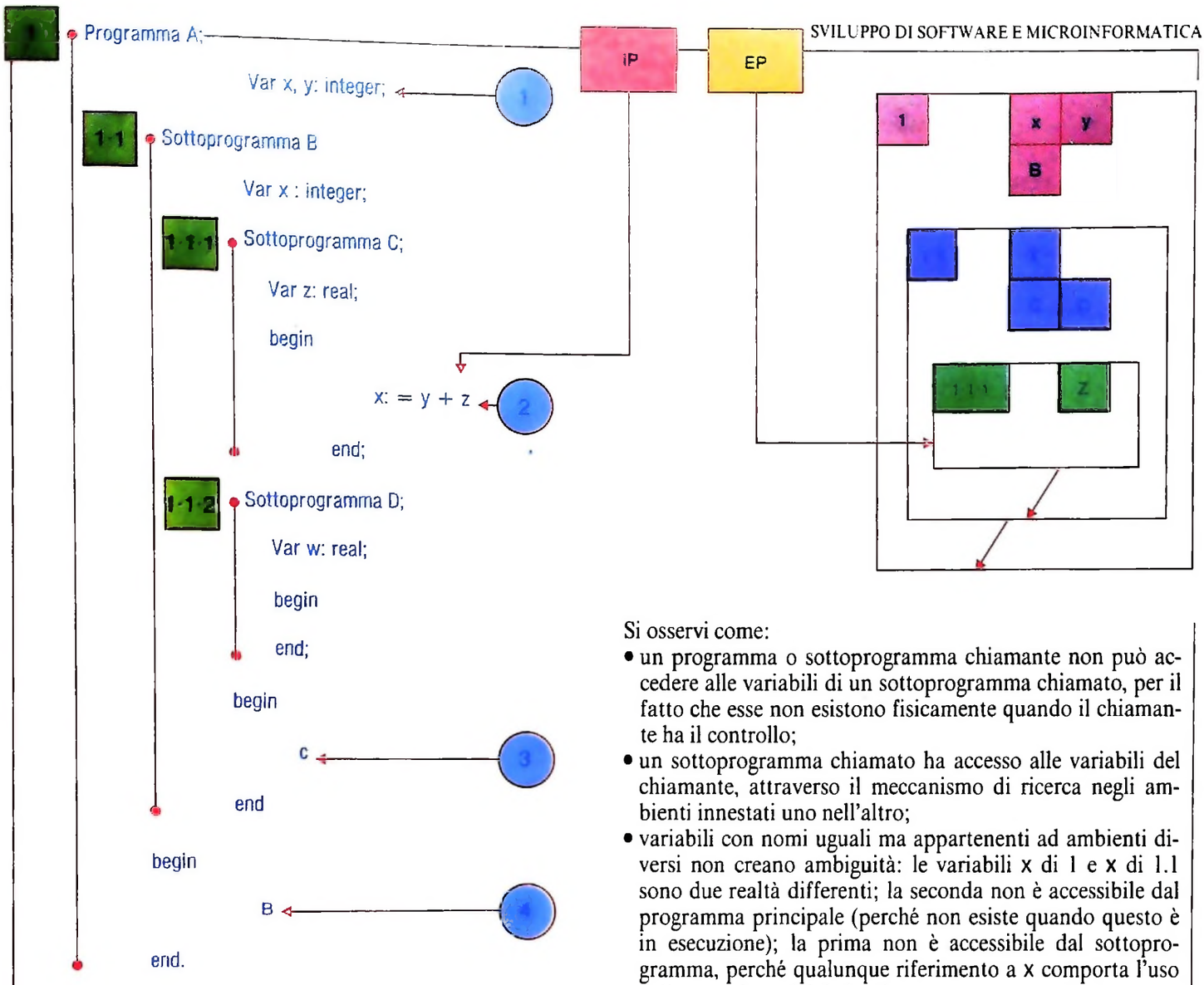
```
VAR k: integer;
```

  o variabili di tipi definiti dal programmatore, come

```
VAR x: giorni;
```

  intendendo con quest'ultima dichiarazione che si vuole avere a disposizione una variabile di nome x, che potrà assumere solo i valori indicati nel tipo "giorno", cioè da 1 a 31.
- Una seconda parte è dedicata alla descrizione delle "risorse" di calcolo, ovvero alla descrizione di eventuali sottoprogrammi richiamati dal programma in questione; è questo l'aspetto più significativo della struttura di un programma; di fatto un sottoprogramma è visto come "risorsa" usabile dal programma né più né meno come una variabile: come una variabile è usata in un assegnamento, e quindi deve essere precedentemente stata dichiarata, così un sottoprogramma può essere richiamato, e anch'esso deve avere avuto una definizione precedente.
- La terza parte del programma è la descrizione dell'algoritmo vero e proprio, con le istruzioni tipiche di assegnamento, di lettura e scrittura, di controllo.

Uno degli aspetti rilevanti di tale organizzazione è che un sottoprogramma ha una struttura assolutamente *identica* a quella di un programma: quindi ciascun sottoprogramma potrà avere le sue variabili, i suoi sottoprogrammi, la sua parte di algoritmo; e la cosa può essere ripetuta senza limiti.



La struttura dinamica di un programma Pascal

Il linguaggio Pascal gestisce *dinamicamente* la memoria: mentre la parte algoritmica resta fissa e invariabile, la parte riguardante le variabili viene creata e distrutta dinamicamente durante l'esecuzione, a seconda delle esigenze.
Vediamone il meccanismo (figura in alto).
Abbiamo un programma A, che usa le variabili x e y e il sottoprogramma B. Quest'ultimo, a sua volta, usa una propria variabile x e due sottoprogrammi C e D, che, rispettivamente, dichiarano le variabili z e w.
L'organizzazione della memoria del programma e dei vari sottoprogrammi durante l'esecuzione (a destra nella figura) è un insieme di *ambienti* (*environment*) in modo che a ogni sottoprogramma corrisponde un proprio ambiente.
Completando ora l'esecuzione di C, s'incontra la END che dichiara la fine del sottoprogramma; ciò causa la *distruzione fisica* dell'ambiente 1.1.1 e il ritorno all'istruzione successiva alla chiamata del sottoprogramma, fatta nell'ambito di B; alla fine di B, viene distrutto l'ambiente 1.1 e l'IP ritornerà alle istruzioni del programma principale.

Si osservi come:
- un programma o sottoprogramma chiamante non può accedere alle variabili di un sottoprogramma chiamato, per il fatto che esse non esistono fisicamente quando il chiamante ha il controllo;
- un sottoprogramma chiamato ha accesso alle variabili del chiamante, attraverso il meccanismo di ricerca negli ambienti innestati uno nell'altro;
- variabili con nomi uguali ma appartenenti ad ambienti diversi non creano ambiguità: le variabili x di 1 e x di 1.1 sono due realtà differenti; la seconda non è accessibile dal programma principale (perché non esiste quando questo è in esecuzione); la prima non è accessibile dal sottoprogramma, perché qualunque riferimento a x comporta l'uso della propria variabile locale.

L'istruzione correntemente eseguita nel programma o nei sottoprogrammi è indicata da un *puntatore di istruzione* (in inglese *instruction pointer*) IP, mentre l'ambiente che in ogni istante è da considerare associato all'istruzione corrente è individuato da un *environment pointer* EP.
Quando l'esecuzione del programma ha inizio, è presente solo l'ambiente individuato dalla cornice 1, con la visibilità e l'accesso alle variabili x e y e al sottoprogramma B (dichiarazioni indicate da 1).
Quando, durante l'esecuzione, IP raggiunge il richiamo del sottoprogramma B (punto 4), si hanno i seguenti effetti:
- trasferimento di IP al sottoprogramma
- allocazione del nuovo ambiente 1.1, con la nuova variabile x ad esso relativa.

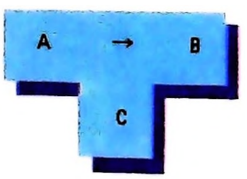
Quando, durante l'esecuzione del sottoprogramma B si raggiunge la chiamata del sottoprogramma C (punto 3), si ottiene il trasferimento di IP a tale sottoprogramma e l'allocazione di un nuovo ambiente 1.1.1, con la relativa allocazione della variabile z.
Quando, durante l'esecuzione di C, IP raggiunge l'istruzione di assegnamento (punto 2), l'assegnamento avviene secondo

le seguenti regole:

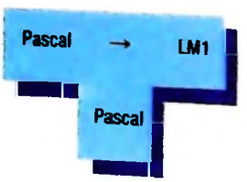
- si cerca y nell'ambiente puntato da EP; non trovandola si verifica se essa è presente nell'ambiente chiamante (1.1); poiché anche qui la variabile non è presente, si risale all'ambiente precedente (1), ove essa viene reperita: se ne prende il valore;
- si cerca nell'ambiente puntato da EP la variabile z; reperitala, se ne somma il valore a quello precedente;
- si cerca quindi nell'ambiente puntato da EP la variabile x; non trovandola, si risale nell'ambiente 1.1, ove una tale variabile viene trovata e ad essa viene assegnato il valore risultante dell'espressione.

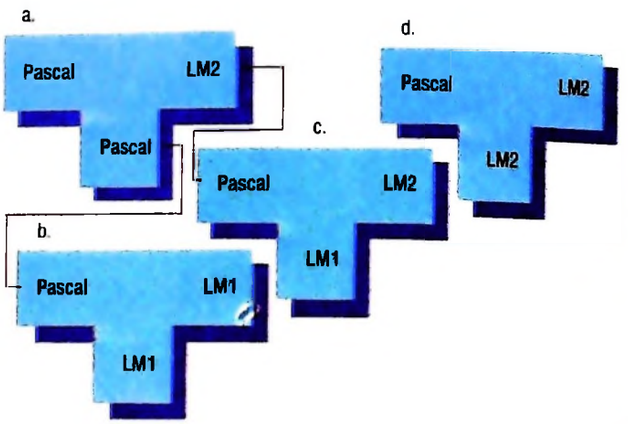
I compilatori Pascal e la tecnica di "bootstrap"

Uno dei motivi della grande diffusione del linguaggio Pascal è legato alla caratteristica con cui sono costruiti i suoi compilatori, scritti... in Pascal!
Anche se la cosa può sembrare un controsenso (infatti, come può essere reso disponibile tale linguaggio, se il suo uso presuppone un compilatore, e se il compilatore è scritto in modo da poter essere usato solo se esso è già disponibile?), di fatto il piccolo sforzo iniziale che ciò impone è ampiamente ripagato dalla riduzione di costi di costruzione dei compilatori successivi.
Consideriamo infatti la seguente situazione: supponiamo di costruire un compilatore che traduca dal linguaggio Pascal al linguaggio proprio di un certo calcolatore, che indichiamo con "Linguaggio Macchina 1" (LM1), e supponiamo di scrivere questo compilatore proprio in Pascal.
Poiché sul calcolatore il linguaggio Pascal non è ancora disponibile, tale compilatore non è in grado di funzionare, essendo scritto in Pascal.
Per poterlo avere a disposizione saremo costretti a "tradurlo a mano" nel linguaggio LM1. A questo punto avremo a disposizione due compilatori Pascal: uno scritto in Pascal e uno scritto nel linguaggio LM1.
Rappresentiamo un compilatore con uno schema come quello illustrato in figura:



Con il diagramma a "T" illustrato intendiamo rappresentare un compilatore scritto nel linguaggio C che, accettando in ingresso un programma scritto nel linguaggio A, lo traduce nel linguaggio B.
Così il nostro compilatore scritto in Pascal, che traduce da Pascal al linguaggio LM1, sarà rappresentato da uno schema come il seguente:





Supponendo di dover costruire un compilatore Pascal per un altro calcolatore, sarà molto comodo partire dal compilatore che traduce da Pascal nel linguaggio LM1 scritto in Pascal, e, alterando opportunamente le parti relative alla traduzione, ottenere un compilatore Pascal scritto in Pascal che traduce in un linguaggio LM2, (blocco a in questo schema operativo).
Avendo a disposizione, per lo sforzo fatto precedentemente, un compilatore Pascal che funziona su un altro calcolatore (blocco b in figura), ed essendo un compilatore Pascal un programma, potremo *compilare* (passo 1 in figura) il compilatore a (che traduce da Pascal a LM2) su questo calcolatore e otterremo (blocco c) un compilatore che traduce da Pascal a LM2, scritto in LM1, e quindi funzionante sul primo calcolatore. Il prodotto risultante è un compilatore Pascal per un calcolatore che può essere usato solo su un altro (si parla in questi casi di cross-compitatore).
Ora compiliamo il compilatore di partenza (cioè il blocco a) con questo cross-compilatore: otterremo un compilatore Pascal che traduce i programmi nel linguaggio ML2, ma che è a sua volta scritto nel linguaggio LM2! Abbiamo a disposizione il compilatore desiderato.
L'unico sforzo che abbiamo dovuto fare nell'operazione è la trasformazione del primo compilatore da Pascal a LM1 nel compilatore da Pascal a LM2, operando con molta semplicità, cioè trasformando un programma in Pascal (il compilatore originale) in uno di poco differente (il blocco a); le restanti operazioni sono semplici compilazioni con programmi già a disposizione.
Questa tecnica riduce lo sforzo di sviluppo di un compilatore anche dell'80%, ed è detta "tecnica di bootstrap", cui è dovuta la larga diffusione dei compilatori Pascal, oltre al fatto che sono stati costruiti buoni compilatori Pascal scritti in Pascal. Uno molto noto è il TRUNK COMPILER (compilatore tronco) di H.H. Naegeli, che, scritto in Pascal, contiene tutte le parti di analisi sintattica di programmi Pascal, di segnalazione di errori, di richiamo di sottoprogrammi di traduzione che però non sono scritti, e che sono lasciati come sottoprogrammi "vuoti", da riempire con le traduzioni opportune a seconda del calcolatore su cui si vuole avere a disposizione il Pascal.

Un po' di grafica (prima parte)

Introduciamo in questa lezione alcuni elementi di grafica che ci permetteranno di costruire immagini sullo schermo del nostro M10.
Iniziamo con un problema preciso: la costruzione di un programma che calcola e traccia le curve del nostro bioritmo.

I bioritmi

Nel corpo umano si verificano un certo numero di fenomeni che presentano andamento periodico: vi sono fenomeni che hanno andamento giornaliero (la temperatura corporea a esempio si alza al tramonto e diminuisce all'alba) e che sono detti CIRCADIANI; fenomeni la cui frequenza è tale che accadono più volte al giorno (si pensi al battito cardiaco), che vengono detti ULTRADIANI; fenomeni che si ripetono a intervalli di più giorni, che vengono detti INFRADIANI. Tra gli ultimi, di un certo interesse sono i cosiddetti BIORITMI. Sembra accertato, infatti, che l'INTELLIGENZA, l'EMOTIVITÀ e lo STATO FISICO siano soggetti a cicli di alti e bassi, con le seguenti durate:

STATO FISICO	23 giorni
EMOTIVITÀ	28 giorni
INTELLIGENZA	33 giorni.

Di fatto, i cultori dei bioritmi sostengono che non sono gli alti e i bassi delle curve relative a determinare problemi di squilibrio, ma il passaggio da un momento di "alto" a uno di "basso" e viceversa; quindi, immaginando di tracciare su un asse orizzontale l'andamento delle varie componenti, ciò che risulta indice di malessere è l'attraversamento di tale asse da parte della curva; anzi, se più curve intersecano l'asse nello stesso momento, il malessere è ulteriormente accentuato.
Di fatto, anche dando completo credito alle affermazioni sopraccennate, le regolarità dei bioritmi sono estremamente difficili da determinare, in quanto il loro periodo non è un fatto assoluto per tutti gli individui (esattamente come non è una norma uguale per tutti la frequenza del battito cardiaco), ma esistono notevoli variabilità, da rendere impraticabile lo strumento se non dopo accuratissime osservazioni che possono durare anni.
Tuttavia, il bioritmo ha conosciuto un certo successo che ci spinge a fornire un programma che ne effettua il calcolo, tanto più che questo sarà un pretesto per l'introduzione di nuove istruzioni BASIC.

Il problema

Vogliamo qui costruire un programma che:
- accetti in ingresso la data di nascita di una persona
- accetti in ingresso l'indicazione di un mese posteriore alla data di nascita precedentemente introdotta
- tracci tre curve relative ai tre ritmi sopraccennati, per il mese indicato, per la per-

sona nata nella data specificata.
Il programma potrà essere organizzato come segue:

- LEGGI LA DATA DI NASCITA
- CALCOLA IL NUMERO DI GIORNI PASSATI DA UNA DATA DI RIFERIMENTO A TALE DATA
- LEGGI IL MESE RICHIESTO PER IL BIORITMO
- CALCOLA IL NUMERO DI GIORNI PASSATI DALLA DATA DI RIFERIMENTO AL PRIMO GIORNO DI TALE MESE
- CALCOLA LA DISTANZA, IN GIORNI, TRA LE DUE DATE, COME DIFFERENZA TRA I DUE NUMERI CALCOLATI PRECEDENTEMENTE
- CALCOLA IL NUMERO DI GIORNI DA CUI È INIZIATO L'ULTIMO CICLO RELATIVO ALLO STATO FISICO:
  - Per fare ciò è sufficiente applicare l'operatore BASIC MOD, che calcola il resto di una divisione; supponiamo infatti che N sia il numero di giorni trascorsi tra le due date interessate; allora, essendo 23 giorni la durata del ciclo per lo stato fisico, dividendo N per 23 (come divisione intera), il quoziente ci fornirà il numero di cicli passati dal momento della nascita all'inizio del mese indicato, mentre il resto della divisione ci indicherà da quanto tempo il nuovo ciclo ha preso inizio; tale operazione, in BASIC, può essere semplicemente indicata con:

    N MOD 23

- ANALOGAMENTE, CALCOLA IL NUMERO DI GIORNI DALL'INIZIO DEL CICLO DELL'EMOTIVITÀ
  - si tratterà in questo caso di fare il calcolo di

    N MOD 28

- ANALOGAMENTE, CALCOLA IL NUMERO DI GIORNI DALL'INIZIO DELL'INTELLIGENZA
  - ancora, con un calcolo del tipo

    N MOD 33

- INFINE, TRACCIA I GRAFICI DEI TRE RITMI.

Iniziamo a costruire la struttura generale del programma, che potrà risultare:

```
10 INPUT"Inserire data di nascita(gg,mm,aa)"
;G,M,A
20 GOSUB 500 'Calcolo n.giorni
30 LET X=N
40 INPUT"Inserire mese richiesto(mm,aa);M,A
50 LET G=1
60 GOSUB 500 'Calcolo n.giorni
70 LET P=N-X
80 CLS
85 'Calcolo dei giorni di inizio dal termine
dell'ultimo
86 'ciclo per ogni ritmo
110 'Traccia i grafici
490 END
500 'Calcola il n. di giorni da una data di
riferimento
```

Il programma così tracciato, nel suo alto livello, individua:

- la lettura della data: giorno, mese e anno nelle tre variabili G, M e A
- il richiamo di un sottoprogramma, che poi ritroviamo all'istruzione 500, che calcola nella variabile N il numero di giorni passati da una data di riferimento a quella fornita
- la memorizzazione del valore N ottenuto dal sottoprogramma, nella variabile X
- la lettura del mese e dell'anno richiesti per il calcolo, nelle variabili M e A
- il posizionamento della variabile G, che rappresenta il giorno di una data, a 1, in modo che la successiva invocazione del sottoprogramma calcoli il numero di giorni passati dalla data di riferimento al primo giorno del mese richiesto
- il calcolo del numero di giorni intercorsi tra le date come differenza tra X e N, memorizzato nella variabile P
- i calcoli successivi, indicati dalle istruzioni di commento.

Poniamoci ora il problema di come effettuare il calcolo richiesto al sottoprogramma alla linea 500.

Di fatto potremmo applicare il seguente algoritmo, che fornisce il risultato per tutte le date successive al 1900:

- calcolare il numero di anni passati come A-1900 in A1
- calcolare in N1 il numero di giorni passati dall'inizio dell'anno fino al mese precedente a quello fornito, tenendo presente che:
  - per M = 1, num. giorni = 0
  - per M = 2, num. giorni = 31
  - per M = 3, num. giorni = 31 + 28 = 59
  - per M = 4, num. giorni = 59 + 31 = 90

e così via fino a dicembre

- calcolare il numero di giorni totale come:

$$N = A1 * 365 + N1 + G$$

- calcolare il numero di anni bisestili intercorsi fino alla data fornita (si tratta di vedere quanti anni divisibili per 4 sono presenti) in B
- calcolare il risultato finale come:

$$N = N + B$$

Di fatto, un comune algoritmo, estremamente efficiente e veloce, reperibile nella letteratura, sfrutta i valori reali e, anche se in modo meno intuitivo, calcola rapidamente il valore interessato; si tratta del seguente programma:

```
510 IF M-3>=0 THEN LET M=M+1:GOTO 530
520 LET A=A-1:LET M=M+13
530 LET N=INT(365.25*A)+INT(30.6*M)+G
540 RETURN
```

A questo punto, siamo già in grado di poter usare il programma, anche se in modo non grafico, e con l'ausilio di qualche intervento manuale: infatti, siamo già in grado di calcolare il numero di giorni da cui ogni ciclo è iniziato, con le seguenti istruzioni:

```
90 LET F=P MOD 23
100 LET S=P MOD 28
105 LET I=P MOD 33
```

e quindi inserire le istruzioni, provvisorie, per la stampa dei risultati:

```
106 PRINT"Fisico:inizio da ";F;" gg."
107 PRINT"Emotiv.:inizio da ";S;" gg."
108 PRINT"Intell.:inizio da ";I;" gg."
```

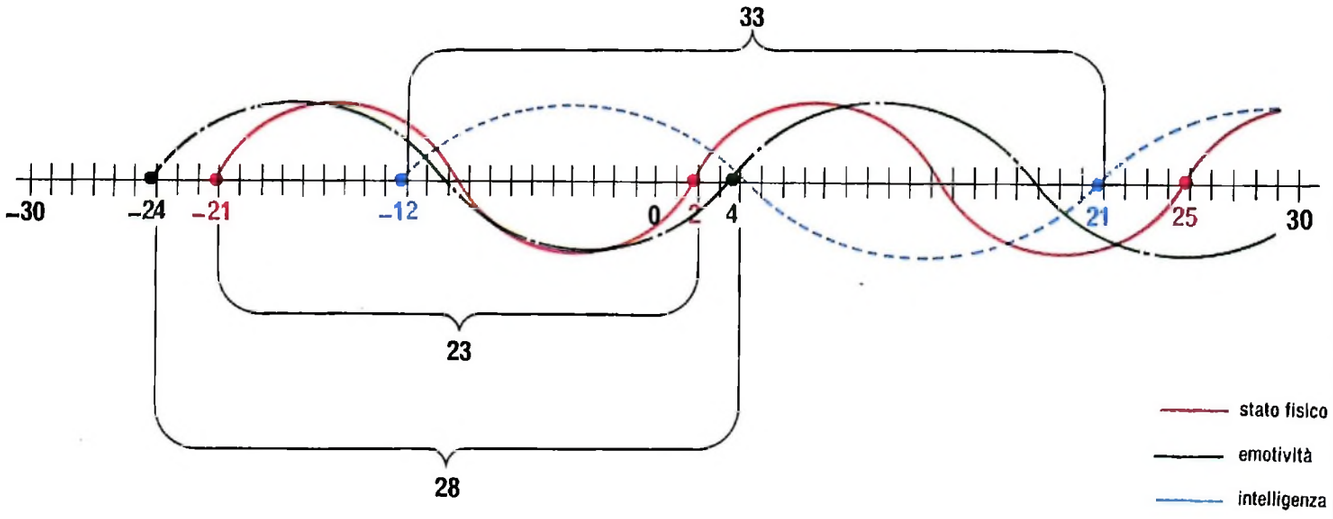
Infatti, eseguendo, otteniamo:

```
run
Inserire data di nascita(gg,mm,aa)? 1,2,
1956
Inserire mese richiesto(mm,aa)? 9,1984
```

otteniamo la schermata:

```
Fisico:inizio da  21  gg.
Emotiv.:inizio da  24  gg.
Intell.:inizio da  12  gg.
Ok
```

A questo punto siamo perfettamente in grado, con carta e matita, di tracciare i grafici che ci interessano, a esempio nel modo seguente:



Vedremo alla lezione successiva come far sì che i grafici vengano tracciati da M10.

Che cosa abbiamo imparato

In questa lezione abbiamo visto:

- l'operatore MOD, che fornisce il resto della divisione intera tra due operandi.

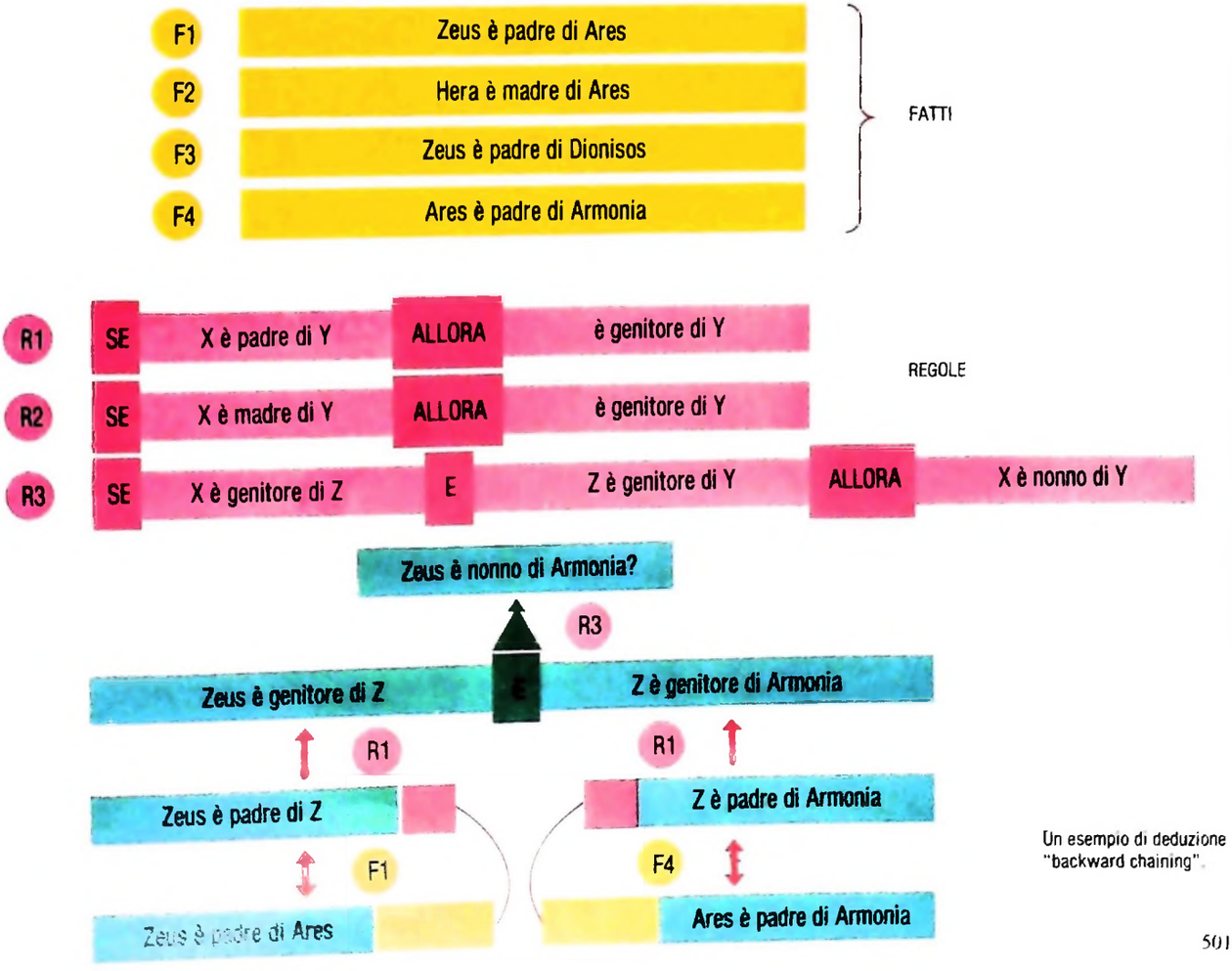
SISTEMI ESPERTI: STRATEGIE DI CONTROLLO

Dopo aver esaminato la struttura e gli scopi di un sistema esperto, vediamo ora come procede nella risoluzione di un problema.

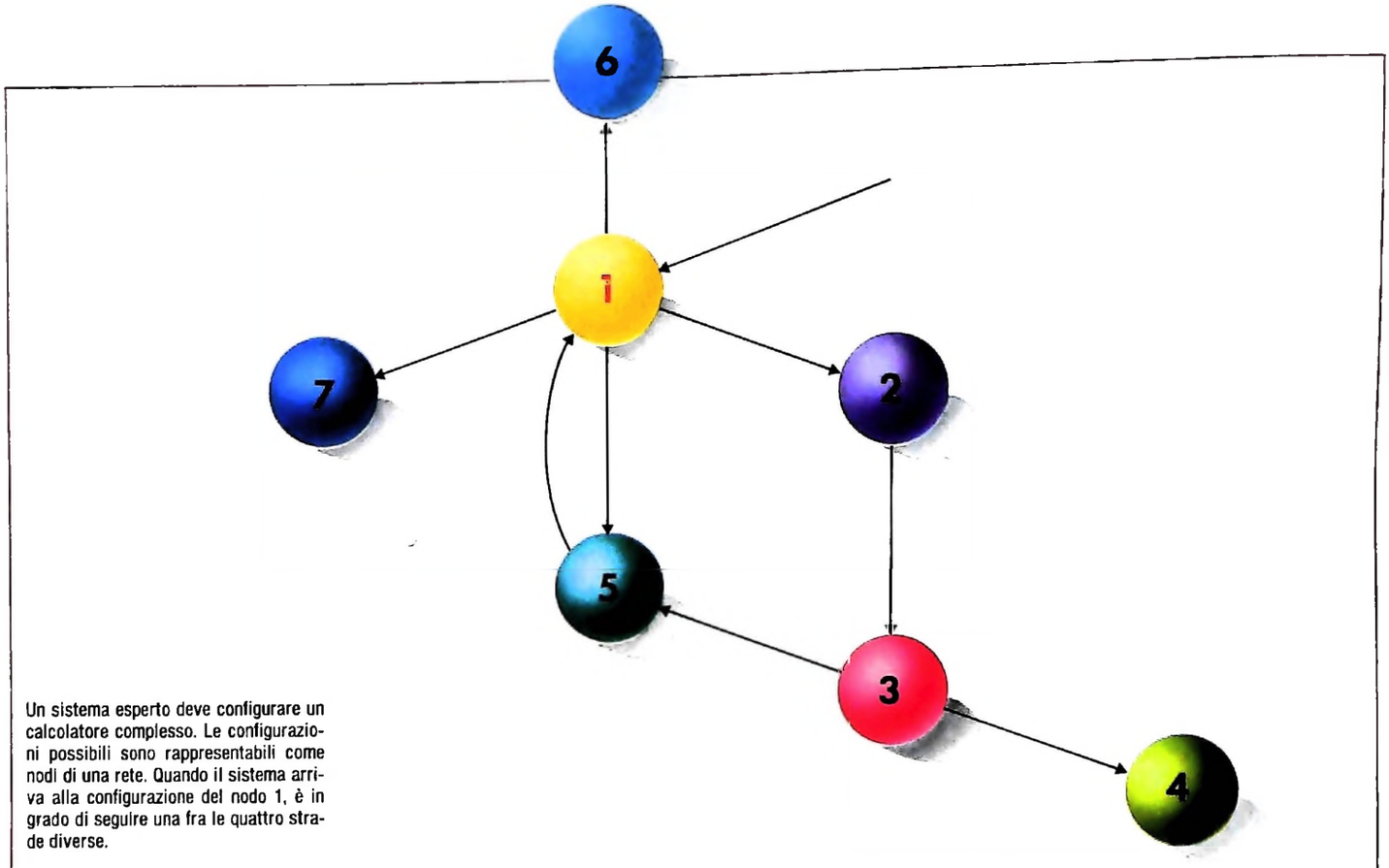
Nel precedente articolo abbiamo parzialmente esaminato la struttura e gli scopi dei sistemi esperti. È possibile isolare nei sistemi esistenti, almeno da un punto di vista logico anche se non sono fisicamente distinti, due componenti, che abbiamo chiamato rispettivamente *base di conoscenza* e *motore di inferenza*. Si è anche detto che la BdC del sistema contiene in generale la conoscenza strettamente relativa al dominio di applicazione. Il motore di inferenza, invece, codifica in sé informazioni più generali sulla strategia di risoluzione di un problema, e può essere più o meno sofisticata anche in dipendenza del modo in cui è strutturata la BdC.

Strategie di controllo

Per dare un esempio di come possa procedere la risoluzione di un problema da parte di un sistema esperto, vediamo brevemente due possibili strategie di controllo, nell'ipotesi di avere una base di conoscenza specifica costituita da regole del tipo illustrato nel precedente articolo (implicazioni SE... ALLORA...). Si vedrà che questi metodi corrispondono a schemi di ragionamento che risultano in parte intuitivi e abbastanza generali; essi sono concretamente adottati in alcuni prototipi esistenti dedicati alla diagnosi di malattie.



Un esempio di deduzione "backward chaining".



Un sistema esperto deve configurare un calcolatore complesso. Le configurazioni possibili sono rappresentabili come nodi di una rete. Quando il sistema arriva alla configurazione del nodo 1, è in grado di seguire una fra le quattro strade diverse.

1) La prima strategia, detta *concatenamento in avanti* (dall'inglese "forward chaining") consiste nel dedurre fatti nuovi (il conseguente di una regola) a partire dai suoi antecedenti quando questi sono verificati. L'aggiunta di questo nuovo fatto alla BdC può rendere vero l'antecedente di un'altra regola, e così via sino a che la catena porta alla derivazione di una risposta significativa. A esempio la regola "SE A presenta un valore inferiore a 250, e B maggiore di 10, ALLORA è plausibile che C sia compreso tra 100 e 200 e D sia positivo", unito ai dati di fatto (che risulteranno a esempio da esplicite richieste all'utente) "A = 185" e "B = 20", porta alla conclusione dei fatti "100 < C < 200" e "D positivo". A questo punto se esiste nella BdC una regola del tipo "SE X è compreso tra 50 e 300, e Y è positivo, ALLORA H deve essere vero", tale regola può scattare, in quanto si è verificato il suo antecedente, il che porta a concludere la verità di H.

2) La seconda strategia, detta *concatenamento all'indietro* (dall'inglese "backward chaining") identifica dapprima una meta per il processo di risoluzione. Verifica quindi quale o quali regole possono portare a questo risultato, e le premesse di queste regole diventano a loro volta delle mete. Il procedimento continua sino a che non si arriva a fatti che sono conosciuti al sistema come veri. A esempio si consideri la regola "SE il soggetto è affetto da X, ALLORA possono manifestarsi gli effetti collaterali Y", unita all'effettiva osservazione, resa nota al sistema, dei sintomi Y. Il sistema stabilirà come meta secondaria del processo di deduzione la verifica del fatto che il paziente sia affetto dalla malattia X, ed esaminerà a ritroso le regole che presentano questo conseguente.

In generale possono esistere più regole che sono contemporaneamente applicabili in un certo istante, che nel nostro caso significa avere diversi antecedenti che sono contemporaneamente verificati (caso 1), o regole con lo stesso conseguente (caso 2). È compito del motore di inferenza scegliere una regola quando si presentano alternative.

Quanto abbiamo visto sinora e soprattutto gli esempi dati presentano necessariamente un quadro molto semplificato della situazione, che può ingannare sulle difficoltà, ma anche sulla potenzialità di questo tipo di approccio nel campo dell'Intelligenza Artificiale. A partire da queste basi, veniamo ad alcuni dei temi principali della discussione sui sistemi esperti. Consideriamo uno di questi sistemi, e in particolare pensiamo, piuttosto che alla sua struttura di programma, all'attività che deve svolgere per affrontare un caso del proprio campo di applicazione. È stato sostenuto da diversi autorevoli studiosi, e anche una riflessione intuitiva lo conferma, che la risoluzione "esperta" di problemi consiste principalmente di *ricerca*. Vediamo che cosa s'intende.

Supponiamo che il nostro sistema debba configurare un complesso calcolatore, cioè produrre degli schemi a partire dai quali è possibile fisicamente assemblare la macchina; gli schemi sono costituiti dall'accostamento di diversi elementi primari, quali memorie, unità centrali, processor aritmetici speciali, bus di connessione e altri. L'ingresso al sistema è costituito da una lista dei componenti desiderati dal cliente finale (l'esempio è stato scelto perché corrisponde all'idealizzazione di un sistema realmente esistente). Questo genere di attività viene svolta normalmente da tecnici particolarmente qualificati che dispongono di approfondite conoscenze tecniche sui componenti da utilizzare e di vasta esperienza.

A un livello di astrazione sufficientemente alto si vede che l'attività del sistema consiste nell'attraversare nel miglior modo possibile uno "spazio" costituito dalle configurazioni ottenibili con i componenti dati. Per avere un'immagine più concreta si può pensare che le configurazioni possibili, anche parziali, siano collegate tra loro da opportuni cammini a formare una specie di rete (che in termini più precisi viene detta *grafo*). Immaginiamo di trovarci in uno dei nodi della rete, che corrisponde a una configurazione parziale: da questo nodo è possibile raggiungerne un altro, corrispondente a una nuova configurazione (che si spera essere più completa della precedente) percorrendo uno qualsiasi dei cammini che si dipartono dal nodo di partenza (figura della pagina a lato). Gli elementi fondamentali da chiarire sono allora i seguenti.

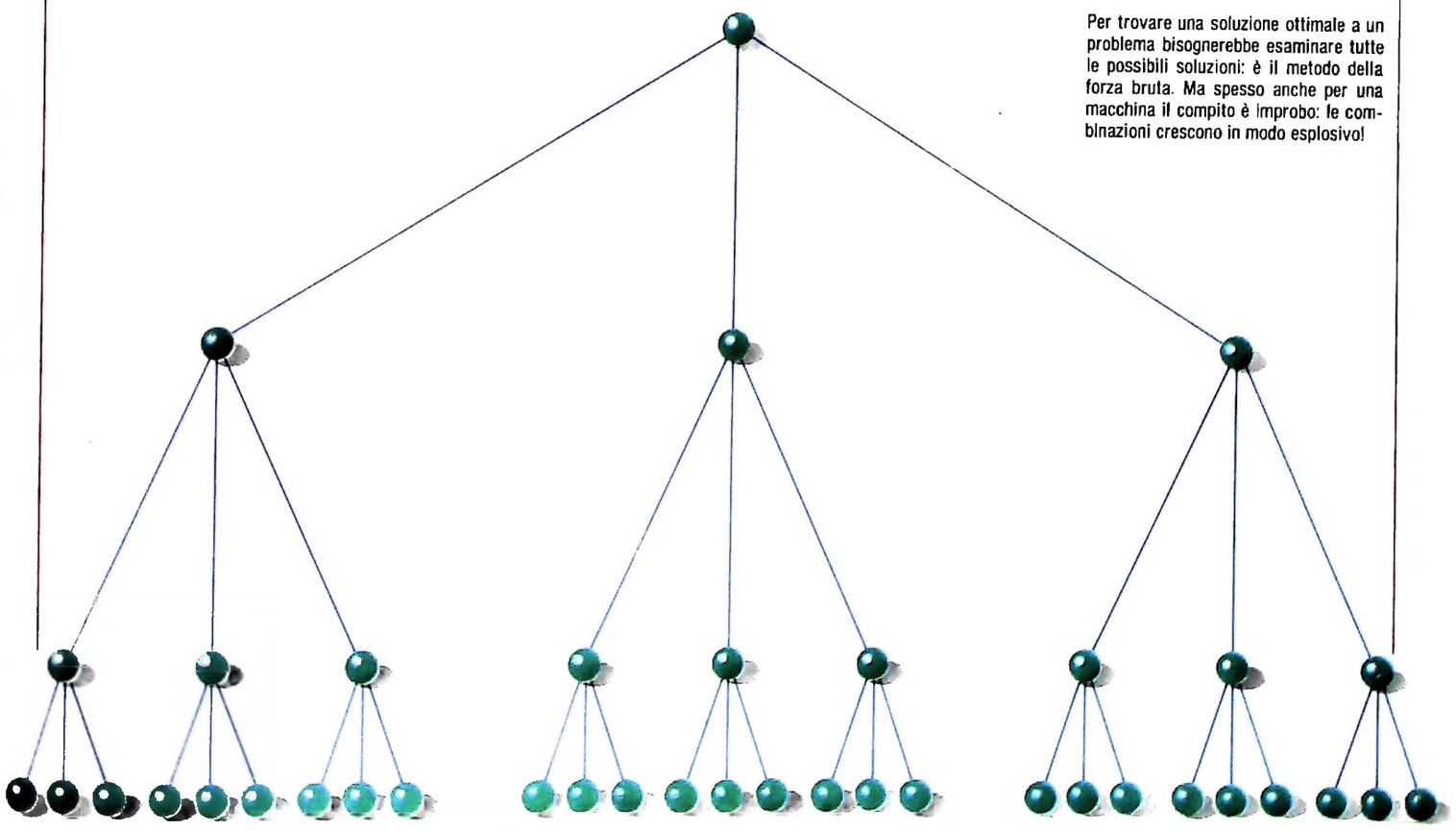
a) *Che cosa significa percorrere un cammino?* Nel nostro caso il sistema disporrà di regole del tipo "SE nell'ordine compare la richiesta di un processor speciale aritmetico e il processor centrale è già stato posizionato, ALLORA sistema il processor aritmetico sul bus nella posizione...". L'esecuzione di questa regola crea una nuova configurazione che contiene anche il processor aritmetico. Percorrere un cammino corrisponde dunque all'esecuzione di una regola. Al di là dell'esempio specifico il discorso è generalizzabile: i nodi non saranno più configurazioni parziali ma rappresenteranno comunque situazioni più o meno vicine alla soluzione, le regole non saranno del tipo considerato, ma unità di conoscenza che si applicano a particolari contesti e ne producono di nuovi, e costituiscono i cammini tra i contesti.

b) *La scelta di una regola*: accade quasi sempre che in particolari contesti possano essere indifferentemente applicate più regole. Abbiamo visto che uno dei compiti del motore di inferenza, probabilmente il più importante, è di scegliere una di queste regole per l'applicazione: sono possibili scelte a diverso livello di sofisticazione che considereremo tra poco. Adottando l'idealizzazione dello spazio di ricerca si vede che una scelta sbagliata può portare fuori strada il percorso complessivo.

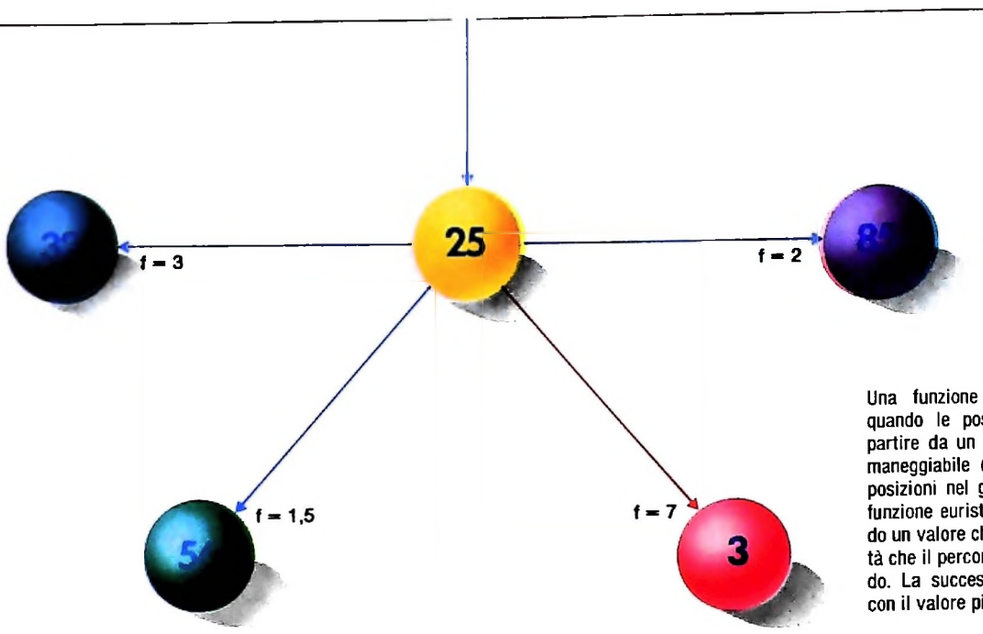
c) *Quando termina la ricerca?* Esplorando lo spazio del problema è possibile perdersi in percorsi che si allontanano dalla soluzione. Questa consiste in un particolare contesto, o insieme di contesti, che devono essere raggiunti limitando al massimo le deviazioni inutili. Naturalmente queste situazioni devono essere riconoscibili dal sistema in modo che a esse non vengano applicate nuove regole e la ricerca possa terminare.

Metodi di ricerca

Questo modo di considerare la risoluzione di un problema permette di chiarire molti punti. In particolare si vede che l'azione di scelta di una regola (o di una unità equivalente,



Per trovare una soluzione ottimale a un problema bisognerebbe esaminare tutte le possibili soluzioni: è il metodo della forza bruta. Ma spesso anche per una macchina il compito è improbo: le combinazioni crescono in modo esplosivo!



Una funzione euristica può aiutare, quando le possibilità aperte sono, a partire da un nodo, in un numero non maneggiabile (miliardi o più, come le posizioni nel gioco degli scacchi). Una funzione euristica produce per ogni nodo un valore che è indice dell'opportunità che il percorso prosegua per quel nodo. La successiva tappa sarà il nodo con il valore più alto.

che potrebbe essere anche una procedura) diventa fondamentale per il comportamento complessivo del sistema: è infatti da essa che dipende la strada percorsa nell'avvicinamento a una soluzione, e perciò il numero e l'entità delle deviazioni inutili rispetto a un percorso ottimale. È naturale quindi che il comportamento del sistema vari grandemente a seconda dei metodi più o meno sofisticati che sono adottati come criterio di scelta. Vediamo alcuni di questi metodi.

La scelta più brutale consiste nel percorrere tutte le strade possibili, cioè applicare ordinatamente tutte le regole legali per un particolare nodo dello spazio. Si comprende immediatamente che l'adozione di questo metodo dipende dal problema trattato, e pochi dei problemi che si presentano concretamente possono essere così affrontati. L'ostacolo è ciò che viene detta "esplosione combinatoriale". Per dare un esempio di cosa si intende, è utile osservare lo spazio della figura della pagina precedente, in cui abbiamo un nodo iniziale e non sono possibili cammini che riportino al punto di partenza. Anche in questo semplice caso, dove da ogni nodo è possibile portarsi in 3 nodi diversi, i possibili cammini, ciascuno costituito da 3 parti, sono $3^3 = 27$. In casi in cui lo spazio sia costituito da miliardi di possibili nodi (come avviene per esempio nel gioco degli scacchi) anche se da ciascun nodo si dipartono pochissime strade, il metodo diventa presto impraticabile fisicamente.

Un possibile aiuto a questa situazione viene dall'uso di una *funzione euristica*. Dopo che tutte le unità di conoscenza che sono valide in un particolare contesto sono state applicate, a ciascuno dei nuovi nodi che sono stati prodotti viene applicata una funzione. Questa funzione produce un valore per il nodo che è un indice dell'opportunità che il percorso sinora seguito prosegua passando per tale nodo (figura in alto). Il nodo con il valore più alto viene quindi scelto come prossima tappa. Questo metodo può permettere un notevole miglioramento nelle prestazioni del sistema: tutto dipende dall'efficacia e dall'affidabilità della funzione euristica, che deve essere applicabile a qualsiasi possibile situazione e avere grandi capacità di "predizione", il che deriva da approfondite analisi del problema e dall'esperienza ricavata dall'implementatore. Data l'importanza delle funzioni euristiche nella ricerca, esse sono state attentamente studiate dal punto di vista delle proprietà matematiche generali che presentano.

Una difficoltà con la funzione euristica è che difficilmente si riesce a darne una forma sufficientemente generale e nello stesso tempo ricca di informazione cosicché possa guidare in modo appropriato la ricerca. Più verosimile è il caso in cui si possiedono criteri di buon senso per la scelta che sono applicabili soltanto in particolari situazioni: questo è equivalente ad avere diverse funzioni euristiche valide soltanto per certi nodi o gruppi di nodi dello spazio del problema. Spesso questi criteri possono venire a loro volta espressi in forma di regole, che sono quindi gestite da un proprio motore di inferenza.

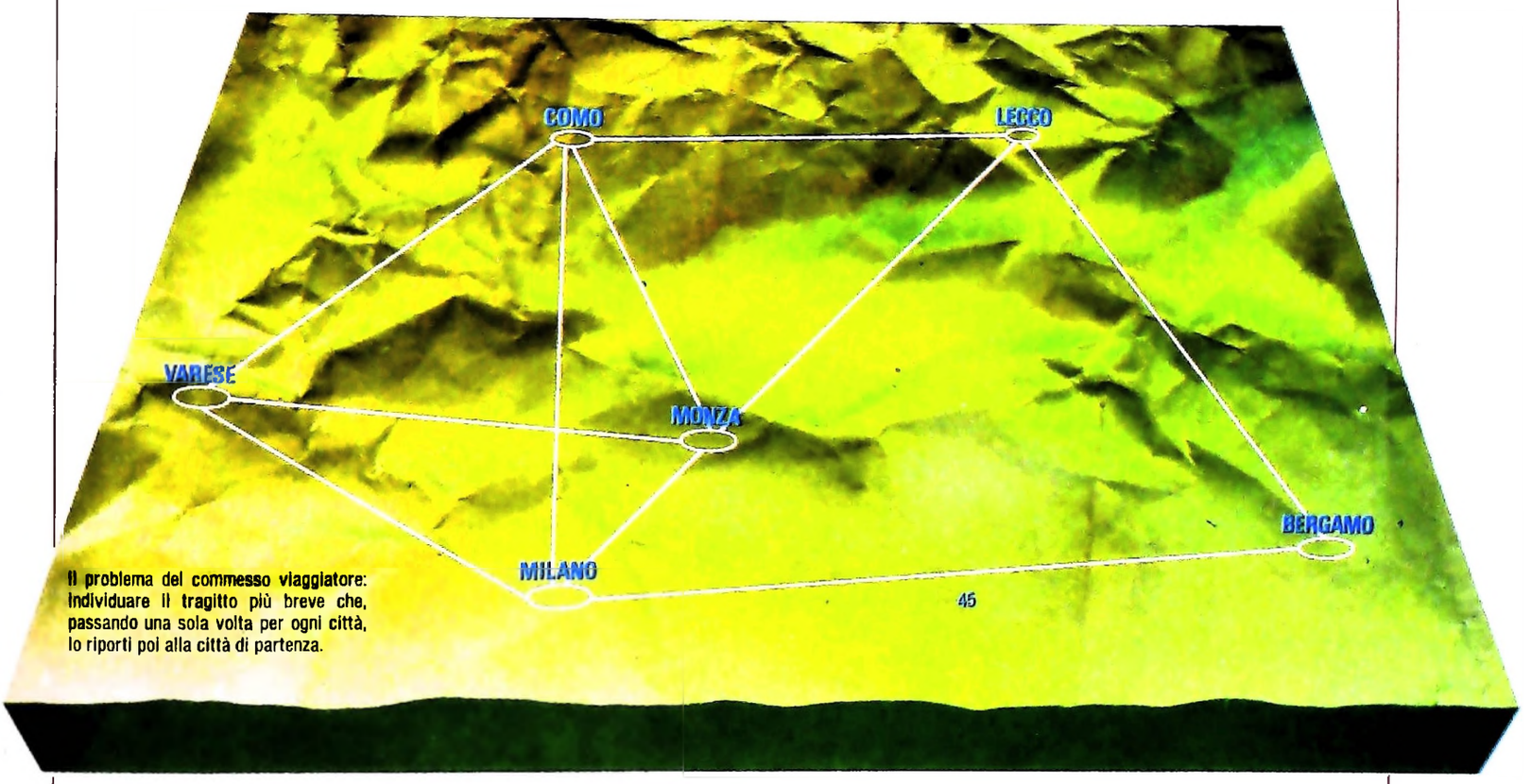
Naturalmente l'efficacia degli ultimi due metodi presentati varia da un sistema all'altro: è possibile che essi talvolta suggeriscano delle strade sbagliate che allontanano localmente dalla soluzione; può quindi essere necessario che il sistema mantenga una traccia dei percorsi seguiti e delle regole applicate, in modo da poter eventualmente ritornare sui passi compiuti.

L'efficacia dei tre metodi illustrati aumenta secondo l'ordine di presentazione: ciò è giustificato dal fatto che è sempre maggiore la conoscenza che viene utilizzata per raggiungere il prossimo nodo (a questo proposito Ed Feigenbaum, uno dei più famosi studiosi di A.I., ha coniato lo slogan: "la conoscenza è potere").

Quanto è stato esposto dovrebbe anche giustificare il fatto che per queste applicazioni è necessario utilizzare macchine particolarmente potenti: il motivo è nella mole di conoscenza che deve essere rappresentata esplicitamente e nella quantità di ricerca che deve essere svolta in tempi ragionevoli per l'interazione con l'utente.

IL PROGETTO DEGLI ALGORITMI

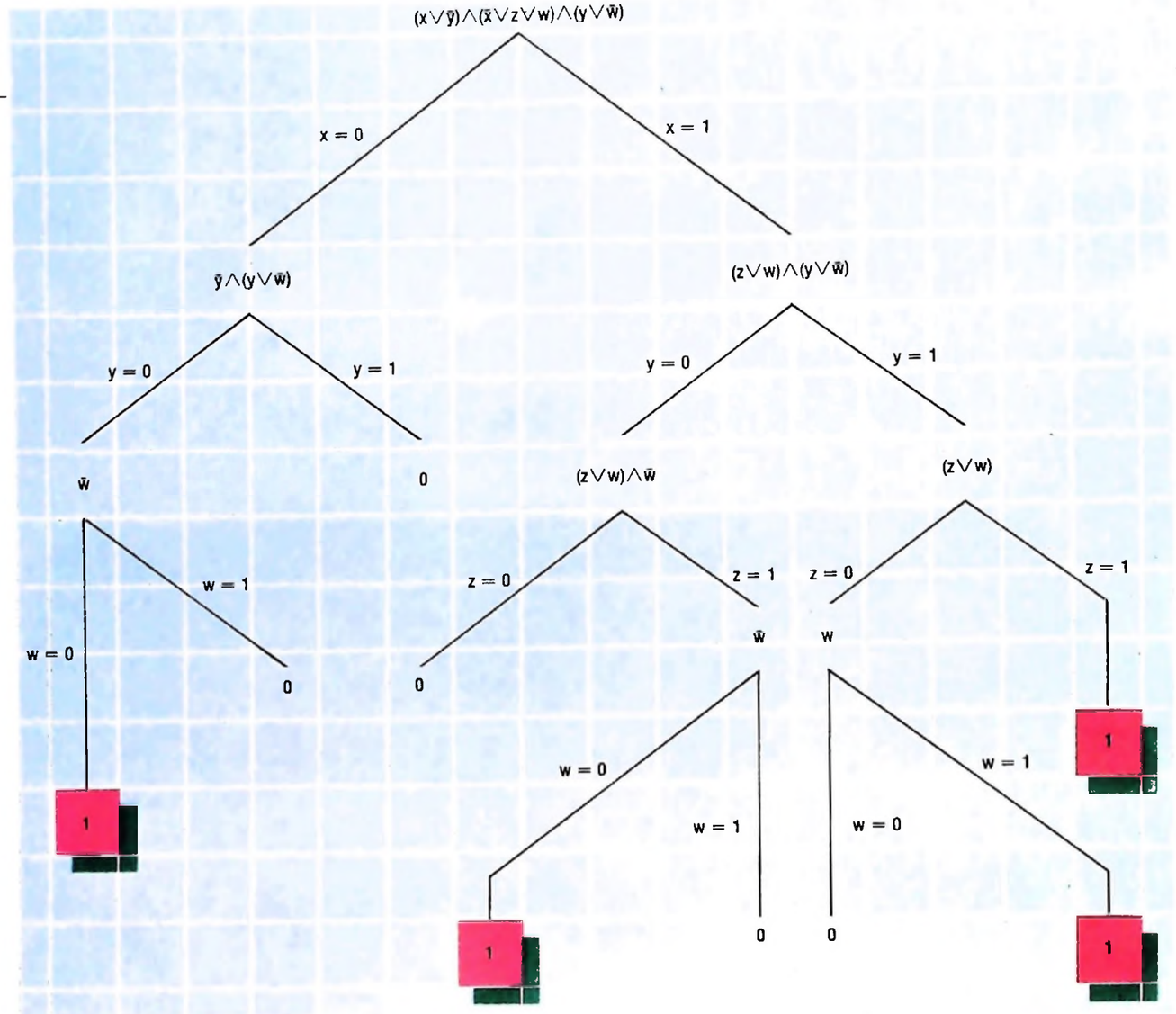
Tecniche generali per costruire algoritmi efficienti applicabili a diversi problemi.



Il problema del commesso viaggiatore: individuare il tragitto più breve che, passando una sola volta per ogni città, lo riporti poi alla città di partenza.

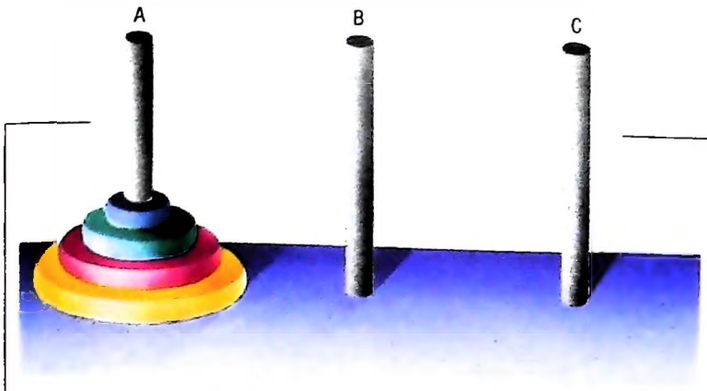
La classe NP

Supponiamo ora di avere un problema P: può verificarsi uno dei seguenti casi:
1) siamo in grado di dimostrare in modo formale che il problema è intrattabile, cioè che la complessità di qualunque algoritmo di soluzione è almeno esponenziale;
2) disponiamo di un algoritmo polinomiale per risolvere P;
3) conosciamo solo algoritmi esponenziali per risolvere P, ma non possiamo escludere che esista un algoritmo polinomiale.
Il primo caso riguarda numerosi problemi di teoria degli automi e dei linguaggi formali e di logica matematica, il problema della raggiungibilità per le reti di Petri e così via. La dimostrazione d'intrattabilità permette di evitare un'inutile ricerca di algoritmi efficienti che non esistono, e di concentrarsi sulla possibilità di risolvere casi particolari.
Nel secondo gruppo di problemi rientrano ad esempio la valutazione dei polinomi, l'ordinamento ecc. Per questi problemi si può, volendo, fare un'analisi più fine, e cercare algoritmi più efficienti di quelli noti, che sono già soddisfacenti: a esempio, per problemi come il prodotto e l'inversione di matrici, importanti nel calcolo numerico, un passaggio dalla complessità cubica dell'algoritmo immediato a una complessità quadratica darebbe notevoli vantaggi pratici.
Resta il terzo gruppo, il più interessante dal punto di vista teorico, che per di più ne contiene moltissimi di notevole rilievo in matematica, logica e via di seguito.
Un esempio classico di questi problemi "di confine" è quello



Un enunciato complesso si dice soddisfacibile se esiste almeno una combinazione di valori di verità delle variabili che contiene che lo rendono vero. Dato che le possibili combinazioni sono 2^n, per provare tutti i casi occorre un tempo esponenziale. Ma procedendo in modo non deterministico, si ottiene un albero di computazione che, con un numero di mosse al più pari al numero delle variabili, consente di decidere se un'espressione e soddisfacibile o meno.

del commesso viaggiatore, cui si riferisce la figura in alto.
Il nostro commesso viaggiatore deve organizzare il proprio giro in un certo numero di città, e conosce la distanza tra due qualsiasi di esse. Il suo obiettivo è trovare il percorso più breve che passi esattamente una volta per ogni città e lo riporti a casa al termine del giro.
Il modo più semplice per arrivare alla soluzione consiste nel costruire tutti i possibili percorsi, scegliendo poi il più conveniente. È però facile vedere che il numero di percorsi possibili cresce esponenzialmente. Infatti, se le città da attraversare sono n, il commesso viaggiatore può scegliere la prima in n−1 modi diversi. Per ognuna di queste possibili scelte, avrà n−2 alternative per la scelta della città successiva e così via. Purtroppo, per questo problema, come per altri dello stesso tipo, non sono noti algoritmi più efficienti di quello esaustivo, ma non se ne può escludere l'esistenza. Perciò, questo gruppo di problemi è oggetto di un'intensa attività di ricerca, con l'obiettivo di risolvere l'ambiguità, collocando in modo preciso ogni problema tra i polinomiali, con la costruzione di un algoritmo efficiente, o tra gli esponenziali, con una dimostrazione di estremo inferiore.
Questa attività di ricerca ha portato a individuare una sottoclasse molto interessante della classe EXP, la classe NP che contiene i problemi che possono essere risolti in tempo polinomiale da una macchina di Turing non deterministica.
Per capire il senso del non determinismo, immaginiamo un uomo in un labirinto. L'uomo può cercare l'uscita esplorando sistematicamente tutte le biforcazioni, tornando indietro fino all'ultima biforcazione non ancora esplorata completamente quando si trova in un vicolo cieco. Questo comportamento è analogo a quello di una macchina di Turing deterministica, che da una configurazione può raggiungere un'unica configurazione successiva, e poi eventualmente tornare indietro per fare altre scelte. Una macchina non deterministica può invece raggiungere, in un certo senso contemporaneamente, un insieme di configurazioni. È come se il nostro uomo nel labirinto a ogni biforcazione potesse, anziché scegliere un'unica strada, creare una copia di se stesso per ogni



Nel gioco della torre di Hanoi, i dischi devono essere spostati uno alla volta da A a B, utilizzando anche l'asta C. Lo spostamento deve essere fatto in modo che un disco non si trovi mai sopra ad uno più piccolo.

strada. Tutte le copie agiscono poi indipendentemente e possono creare nuove copie; la prima copia che trova l'uscita comunica il successo alle altre che scompaiono.
In sostanza, un problema è NP quando la sua complessità dipende non tanto dalle singole soluzioni possibili quanto dal numero elevato di queste possibili soluzioni.

I problemi NP completi

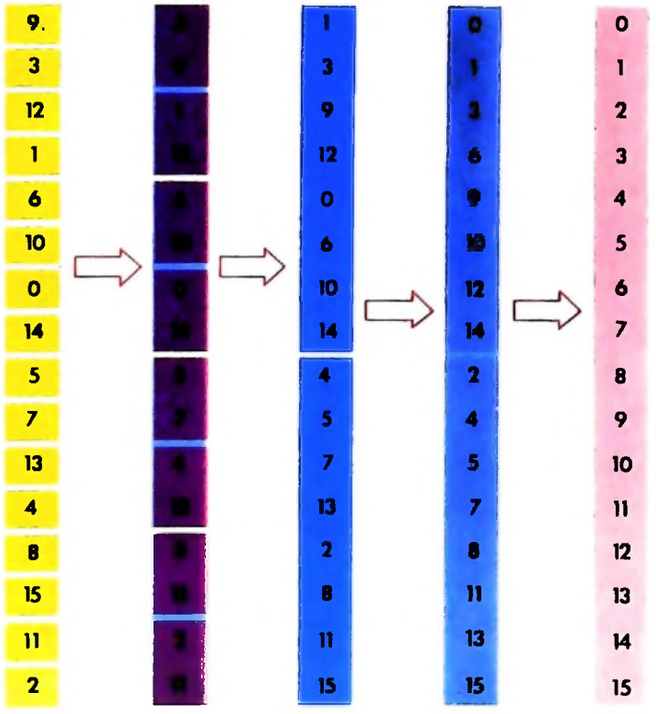
A questo punto, ci si può chiedere se c'è qualche problema che appartenga alla classe NP, ma sicuramente non alla classe P: la risposta è ignota, e questa è un'altra delle grandi questioni irrisolte nella teoria della complessità.
L'uso della nozione di riducibilità polinomiale ha consentito di isolare, all'interno della classe NP, una sottoclasse costituita dai problemi per così dire "più difficili" in NP. Questi problemi si chiamano NP-completi. Formalmente, un problema è NP-completo se appartiene a NP e inoltre ogni altro problema in NP è polinomialmente riducibile a esso.
L'importanza della classe dei problemi NP-completi è evidente: basterebbe trovare un algoritmo polinomiale per un solo problema NP-completo per riuscire a risolvere in tempo polinomiale, con il meccanismo della riduzione, tutti i problemi in NP; così la classe NP coinciderebbe con P.
Oggi sono note alcune centinaia di problemi NP completi, tra cui il problema del commesso viaggiatore e il problema della soddisfacibilità delle espressioni booleane (enunciati), il primo scoperto, presentato nella figura della pagina a lato.

Come costruire algoritmi efficienti

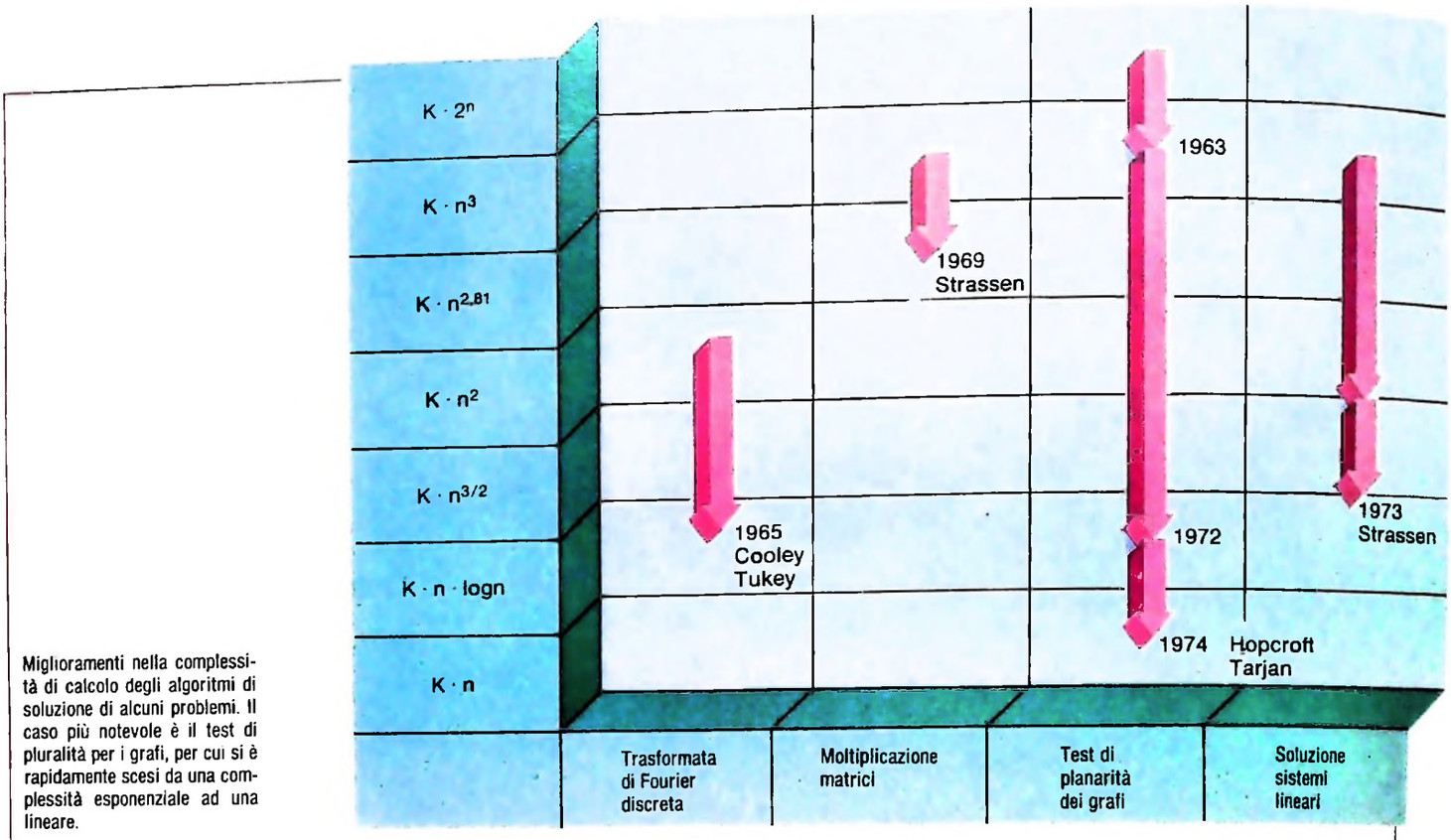
Dopo aver visto come si arriva a stabilire risultati negativi, che cioè escludono l'esistenza di algoritmi efficienti, occupiamoci ora brevemente delle tecniche per fornire risultati positivi, cioè per costruire algoritmi efficienti.
Spesso, la costruzione di un algoritmo efficiente è legata a un'intuizione "geniale", oppure a particolarità dei dati del problema, e non è quindi trasferibile ad altri problemi.
Tuttavia, prendono corpo alcune tecniche generali per il progetto di algoritmi efficienti, applicabili a diversi problemi. Ne considereremo due: la tecnica del "divide et impera" e la programmazione dinamica.

"Divide et impera"

Il nome esprime bene il meccanismo su cui si basa questa tecnica: suddividere il problema in tanti sottoproblemi più piccoli per ridurne la complessità.
Come esempio, possiamo considerare il gioco della torre di Hanoi. Ci sono tre aste di legno, A, B e C, nella prima delle quali sono infilati alcuni dischi di diametro decrescente dal basso verso l'alto (figura a lato). Si chiede di spostare i dischi uno alla volta fino ad averli tutti infilati nell'asta B nello stesso ordine, e in modo che in nessuno dei passaggi intermedi qualche disco venga a trovarsi sopra a uno più piccolo.
La soluzione con la tecnica del "divide et impera" è ottenuta riducendo il problema di spostare gli n dischi più piccoli da A a B a due sottoproblemi di dimensione n-1:
— spostare i primi n-1 dischi da A a C;
— spostare l'ennesimo disco da A in B;
— spostare gli n-1 dischi infilati in C in B.
A sua volta, lo spostamento di n-1 dischi viene ridotto al problema di spostare n-2 dischi, e così via, fino a ridurre tutto allo spostamento di dischi singoli. È difficile visualizzare e simulare a mano gli spostamenti effettivi dei singoli dischi, ma il funzionamento dell'algoritmo è facile da capire, e la sua correttezza è evidente, dato che ogni sottoproblema è molto ben delimitato, e non si coinvolge un disco finché non sono stati sistemati tutti quelli più piccoli.
Per quanto riguarda l'efficienza, l'algoritmo visto non è par-



Per ordinare 16 numeri, l'algoritmo di merge-sort richiama l'ordinamento dei primi otto e degli otto successivi, che poi "immerge" in un basso numero di confronti. I due gruppi di otto numeri sono a loro volta ricondotti all'ordinamento di quattro gruppi di quattro e quindi a otto gruppi di due: ne consegue che per ordinare 1000 numeri sono sufficienti circa 10000 confronti contro il milione richiesto invece dall'algoritmo lineare.



Miglioramenti nella complessità di calcolo degli algoritmi di soluzione di alcuni problemi. Il caso più notevole è il test di pluralità per i grafi, per cui si è rapidamente scesi da una complessità esponenziale ad una lineare.

ticolarmente conveniente, perché la dimensione dei sottoproblemi non differisce molto da quella del problema iniziale. In molti casi però è possibile ridurre la soluzione di un problema di dimensione n alla soluzione di due sottoproblemi di dimensione n-2: in questi casi l'algoritmo è molto più efficiente di algoritmi più diretti.

Un esempio molto importante è il problema dell'ordinamento. Un algoritmo immediato, detto di selezione lineare, consiste nello scegliere il più piccolo tra gli n elementi dati, poi il più piccolo tra gli n-1 rimanenti e così via. Si vede facilmente che questo algoritmo richiede un numero di confronti proporzionale a n^2.

Un algoritmo più efficiente è il merge-sort, basato sulla tecnica "divide et impera". Il funzionamento di questo algoritmo è descritto nella figura della pagina precedente, in basso. Si osservi anche che merge-sort è un algoritmo ottimo per l'ordinamento, poiché la complessità di questo problema non può essere inferiore a $n \cdot \log n$.

La programmazione dinamica

Spesso non è possibile ridurre un problema a un piccolo numero di sottoproblemi di dimensione molto inferiore: a esempio, potremmo arrivare solo a una suddivisione in n sottoproblemi di dimensione n-1. In questo caso, procedendo nello stesso modo, genereremmo n-1 sottoproblemi di dimensione n-2 per ognuno degli n sottoproblemi di dimensione n-1, e così via, e dovremmo quindi affrontare un numero di sottoproblemi che cresce esponenzialmente.

Fortunatamente, in molti casi la situazione non è grave come sembra. Infatti, capita spesso che i sottoproblemi generati in numero esponenziale non siano in realtà tutti distinti, ma che vi sia un numero polinomiale di sottoproblemi ognuno dei quali è generato più volte. Dopo aver risolto una volta un sottoproblema, possiamo allora memorizzare la soluzione in una tabella, in modo che se lo stesso sottoproblema si presenta in seguito non dobbiamo più risolverlo, ma ci basta cercare la soluzione nella tabella.

Le tecniche di risoluzione di problemi attraverso la costruzione di tabelle di questo tipo vanno sotto il nome di programmazione dinamica.

L'impiego di tecniche come il "divide et impera" e la programmazione dinamica ha consentito di abbassare, anche in modo rilevante, l'estremo superiore di complessità per molti problemi. La figura in alto riporta, a titolo di esemplificazione, i miglioramenti ottenuti per alcuni importanti problemi.

Per il calcolo della trasformata di Fourier discreta, Cooley e Tukey hanno fornito nel 1965 un algoritmo basato su tecniche ricorsive che abbassa il tempo di calcolo da n^2 a $n \cdot \log n$.

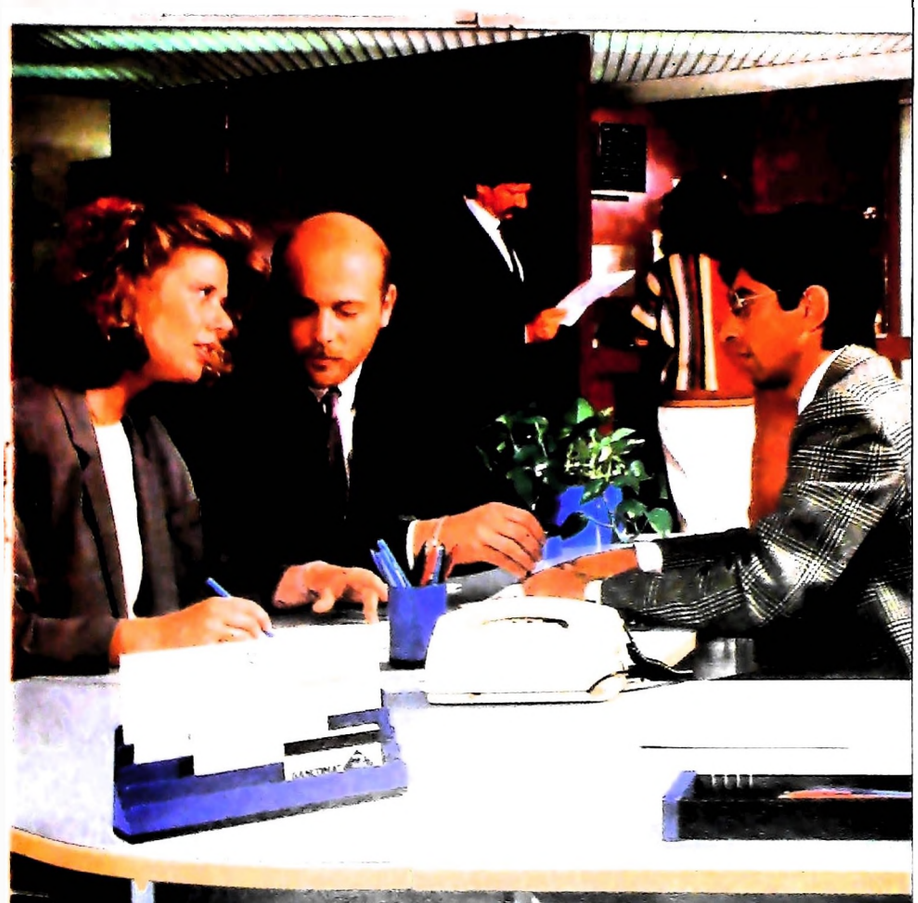
Un secondo esempio interessante è il prodotto di matrici, per cui si è scesi, con l'algoritmo di Strassen che è un eccellente esempio di "divide et impera", da n^3 a $n^{2.81}$.

Un miglioramento analogo si è avuto per la soluzione di sistemi di equazioni lineari, problema che è strettamente collegato al prodotto di matrici.

Ancora più notevoli sono stati i progressi per il test di planarità dei grafi, problema di enorme rilevanza pratica per il progetto di circuiti. In questo caso, una serie di miglioramenti successivi ha consentito di passare dal tempo esponenziale dell'algoritmo più immediato al tempo lineare di Hopcroft e Tarjan del 1974.

—UN NUOVO MODO DI USARE LA BANCA.—

Conto corrente più



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

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

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

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

Inoltre un servizio utilissimo, soprattutto per imprenditori e commercianti denominato "esito incassi", consente di avere comunicazione dell'eventuale insolvenza entro solo cinque giorni dalla scadenza. Una opportunità veramente speciale.

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

Veniteci a trovare, ci conosceremo meglio.

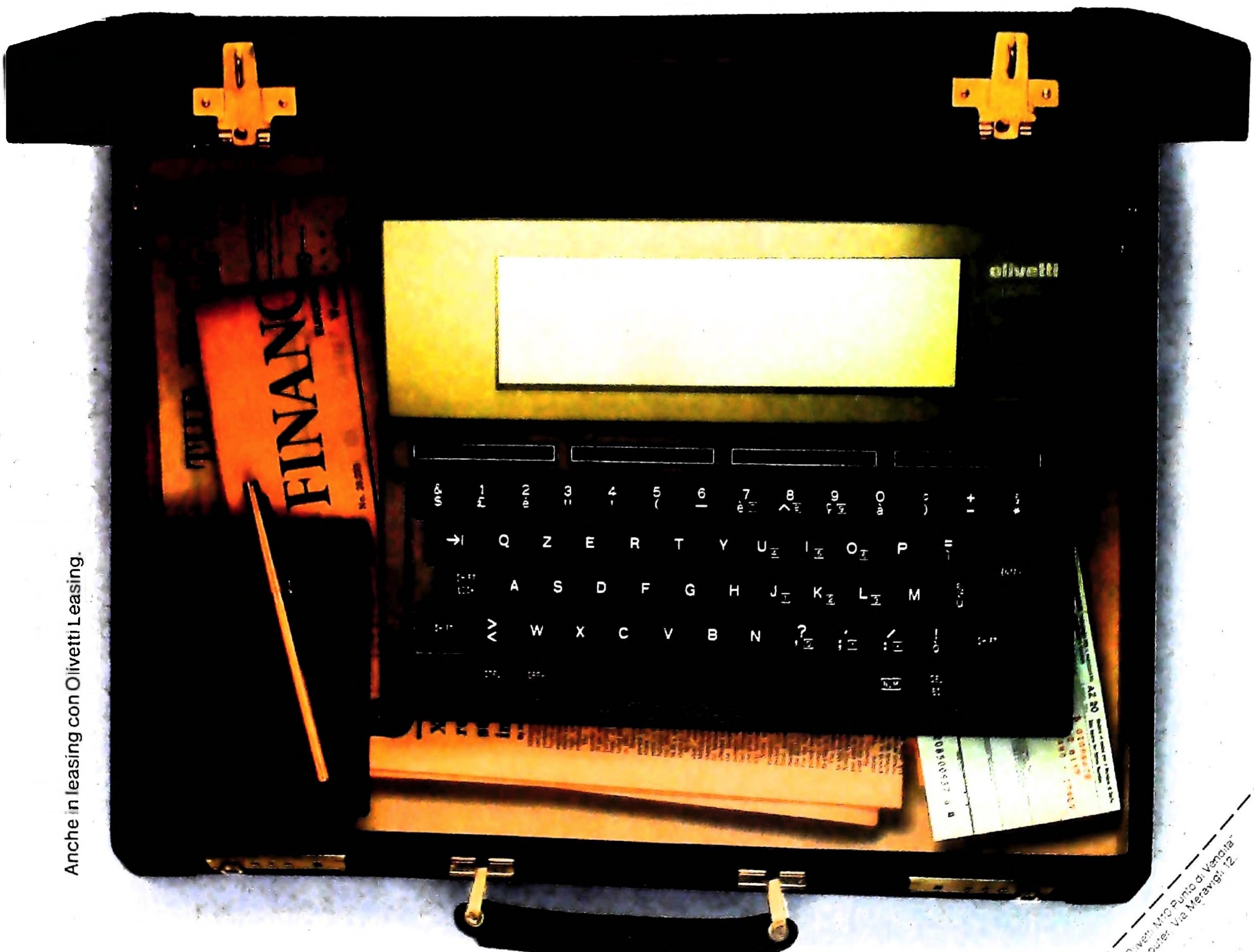
BANCO DI ROMA
CONOSCIAMOCI MEGLIO.

Olivetti M10 vuol dire disporre del proprio ufficio in una ventiquattrore. Perché M10 non solo produce, elabora, stampa e memorizza dati, testi e disegni, ma è anche capace di comunicare via telefono per spedire e ricevere informazioni. In grado di funzionare a batteria oppure collegato all'impianto elettrico, M10 mette ovunque a disposizione la sua potenza di memoria, il suo display orientabile a cristalli liquidi capace anche di elaborazioni grafiche, la sua tastiera professionale arricchita da 16 tasti funzione.

Ma M10 può utilizzare piccole periferiche portatili che ne ampliano ancora le capacità, come il microplotter per scrivere e disegnare a 4 colori, o il registratore a cassette per registrare dati e testi, o il lettore di codici a barre. E in ufficio può essere collegato con macchine per scrivere elettroniche, con computer, con stampanti. Qualunque professione sia la vostra, M10 è in grado, dovunque vi troviate, di offrirvi delle capacità di soluzione che sono davvero molto grandi. M10: il più piccolo di una grande famiglia di personal.

PERSONAL COMPUTER OLIVETTI M10

L'UFFICIO DA VIAGGIO



Anche in leasing con Olivetti Leasing.

a negozi contrassegnati da "Olivetti M10 Punto di Vendita"
coupon a Olivetti Divisione Personal Computer, Via Meravigli 12
NOME COGNOME
VIA N
CAP CITTÀ
TELEFONO

olivetti