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

44 CORSO PRATICO COL COMPUTER

421966

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

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
MARCO ANELLI, DIEGO BIASI, ANDREA GRANELLI, ALDO GRASSO, MARCO MAIOCCHI, DANIELE MARINI, GIANCARLO MAURI, CLAUDIO PARMELLI

Testi
VIRGINIO SALA, ADRIANO DE LUCA, DOMENICO CAVALLOTTO, 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

LE MACCHINE A STATI FINITI

Una macchina automatica che distribuisce caffè ci offre la possibilità di studiare una semplice struttura astratta che simula dispositivi di calcolo simili ai calcolatori digitali.

Vi sarà sicuramente capitato almeno una volta di vedere una di quelle macchine automatiche che distribuiscono caffè, bevande, biglietti del tram o qualche altro tipo di oggetto, se vi si inserisce un gettone oppure una determinata quantità di monete. Una delle più semplici è la macchina che fornisce caffè amaro o zuccherato se si inserisce nell'apposita fessura un gettone e si preme uno dei due pulsanti disponibili: un tipo di macchina che si può trovare abbastanza spesso nei grandi uffici.

Una macchina di questo genere si presta ad essere analizzata in modo astratto come una *macchina a stati finiti*: anche un calcolatore digitale è una macchina a stati finiti.

La nostra macchina distributrice di caffè ha solo tre stati possibili: rimane in uno stato di attesa fino a quando non viene introdotto un gettone e premuto un pulsante.

A seconda del pulsante che viene premuto dopo aver introdotto il gettone, la macchina passa dallo stato di attesa nello stato "emissione di caffè amaro" oppure nello stato "emissione di caffè zuccherato".

Dallo "stato di emissione", la distributrice di caffè ritorna poi ancora nello stato di attesa, fino al momento in cui arriva un'altra persona che inserisce un gettone e preme uno dei due pulsanti.

L'attività della macchina ha due possibili risultati, una tazza di caffè dolce o una tazza di caffè zuccherato, in risposta a due possibili tipi di input, costituiti da altrettante coppie gettone-pulsante. Per comodità, possiamo supporre che il pulsante per il caffè dolce sia rosso e quello per il caffè amaro sia verde.

La macchina in questo momento è inattiva: è nello stato di attesa, chiamiamolo S_0. Arriva il signor Rossi, inserisce un gettone e preme il pulsante rosso: la macchina passa nello stato S_1 ed emette caffè dolce. Dopodiché torna allo stato S_0 di attesa.

Arriva il signor Bianchi, inserisce un gettone e preme il bottone verde: la macchina passa nello stato S_2, emette caffè amaro, poi ritorna allo stato S_0. E via di seguito.

Possiamo pensare che tutta l'attività della macchina sia scandita da una sorta di orologio il quale segna "istanti" di tempo: t_1, t_2, t_3 e via dicendo; chiamiamo t_0 lo stato iniziale.

L'attività della macchina, per l'esempio appena fatto, può essere descritta temporalmente come nella tabella riportata a pagina seguente.

Le caratteristiche di una macchina a stati finiti

L'esempio della macchina distributrice di caffè ci permette di evidenziare alcune caratteristiche fondamentali di una macchina a stati finiti.

Innanzitutto, la sua sincronizzazione: l'attività della macchina è regolata, come già accennato, da impulsi temporali discreti, da una sorta di orologio. Può succedere qualcosa solamente negli istanti definiti dagli impulsi; fra un impulso e l'altro possiamo immaginare che la situazione resti, per così dire, "congelata".

In secondo luogo, la macchina è deterministica: sussiste un rapporto ben preciso fra ciascun input, o ciascuna successione di input, e il comportamento della macchina. Se inserisco un gettone e premo il pulsante rosso, ottengo caffè dolce; non c'è possibilità che dalla macchina esca casualmente una volta un caffè, una volta una pera e la volta successiva l'ultima edizione del quotidiano locale.

Il comportamento della macchina è stabile, e non sono in gioco né la casualità, né la probabilità, né qualche elemento magico. (La macchina può sempre guastarsi, ma il mondo delle analisi astratte è sempre molto ottimistico e non tiene conto di questa possibilità.)

La macchina, in terzo luogo, accetta qualcosa in input: gli elementi di ingresso sono ben definiti e a ciascun ingresso fra quelli "legittimi" la macchina risponde in qualche modo. Ogni altro ingresso viene ignorato. (Se nella macchina distributrice di caffè venisse inserita una moneta da 100 lire al posto dell'apposito gettone, verrebbe "mangiata" senza alcun risultato.)

La macchina può disporsi in un numero finito di stati per cui in ogni istante ha la possibilità di trovarsi in uno solo di questi stati.

Lo stato in cui la macchina si disporrà all'istante successivo è in dipendenza sia dello stato in cui si trova, sia dell'input che riceve.

Infine, la macchina ha un output o un certo insieme di possibili output o uscite. Ciò che esce dalla macchina a un certo istante dipende dallo stato in cui si trova la macchina (e di conseguenza dall'input che ha ricevuto).

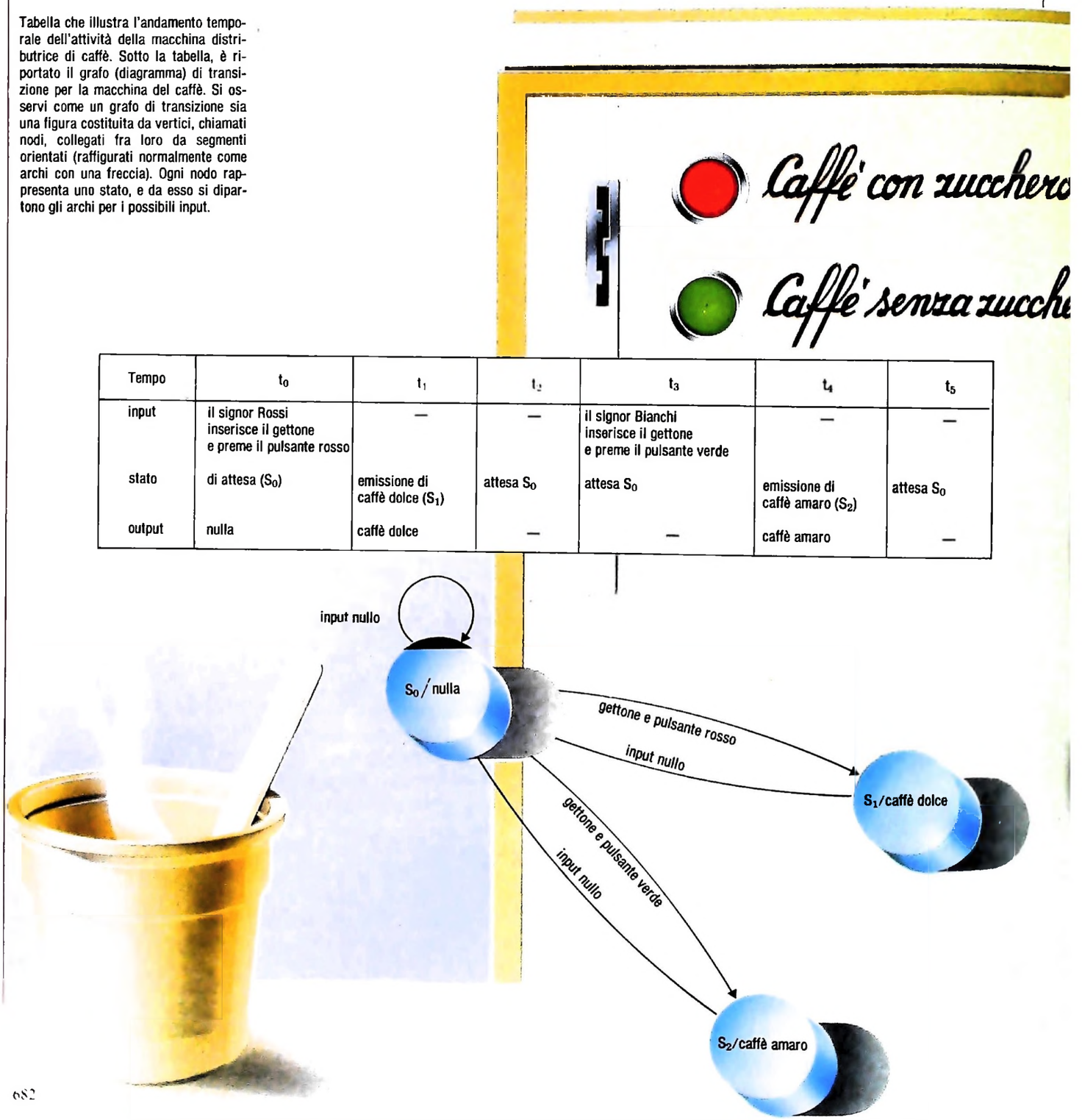
Va notato che, grazie al fatto di passare in un certo stato in funzione dell'input ricevuto, la macchina ha, almeno in una certa misura, una "memoria" dell'input.

Una definizione astratta

Possiamo riassumere quanto abbiamo fin ora detto in una espressione matematica precisa e astratta: una macchina a stati finiti M è una struttura $<S, I, O, f_s, f_o>$, dove S è un insieme finito di stati, I è un insieme finito di simboli di input (l'alfabeto di input), O è un insieme finito di simboli di output (l'alfabeto di output), f_s è una funzione che associa uno stato a ogni coppia costituita da uno stato e da un simbolo di ingresso, f_o è una funzione che a ogni stato fa corrispondere un simbolo di output.

La funzione f_s: $S \times I \rightarrow S$ è la funzione "stato successivo": determina quale sarà il prossimo stato della macchina, in funzione dello stato attuale e dell'input attuale. f_o è la funzione di output e stabilisce l'output corrispondente per ciascuno stato in cui la macchina può trovarsi.
Il modo più efficace per dare le funzioni di stato successivo e di output è quello di presentarle sotto forma di tabella: la tabella a sinistra della pagina accanto dà un esempio di funzioni f_s e f_o per una macchina a stati finiti con tre stati S_0, S_1 e S_2, due input (0 e 1) e due output (0 e 1). Nella tabella a destra (pagina accanto) viene presentato l'andamento tempora-

Tabella che illustra l'andamento temporale dell'attività della macchina distributrice di caffè. Sotto la tabella, è riportato il grafo (diagramma) di transizione per la macchina del caffè. Si osservi come un grafo di transizione sia una figura costituita da vertici, chiamati nodi, collegati fra loro da segmenti orientati (raffigurati normalmente come archi con una freccia). Ogni nodo rappresenta uno stato, e da esso si dipartono gli archi per i possibili input.



Tempo	t_0	t_1	t_2	t_3	t_4	t_5
input	il signor Rossi inserisce il gettone e preme il pulsante rosso	—	—	il signor Bianchi inserisce il gettone e preme il pulsante verde	—	—
stato	di attesa (S_0)	emissione di caffè dolce (S_1)	attesa S_0	attesa S_0	emissione di caffè amaro (S_2)	attesa S_0
output	nulla	caffè dolce	—	—	caffè amaro	—

stato attuale	stato successivo		output
	input 0	input 1	
S_0	S_1	S_0	0
S_1	S_2	S_1	1
S_2	S_2	S_0	1

La tabella a sinistra dà un esempio di macchina a stati finiti con tre stati (S_0, S_1 e S_2), due input e due output. La tabella sottostante ci mostra, nella stessa macchina, una sequenza di input consistente nei caratteri 01011 (leggendo da sinistra a destra).

tempo	t_0	t_1	t_2	t_3	t_4	t_5
input	0	1	0	1	1	—
stato	S_0	S_1	S_2	S_0	S_0	S_0
output	0	1	1	0	0	0

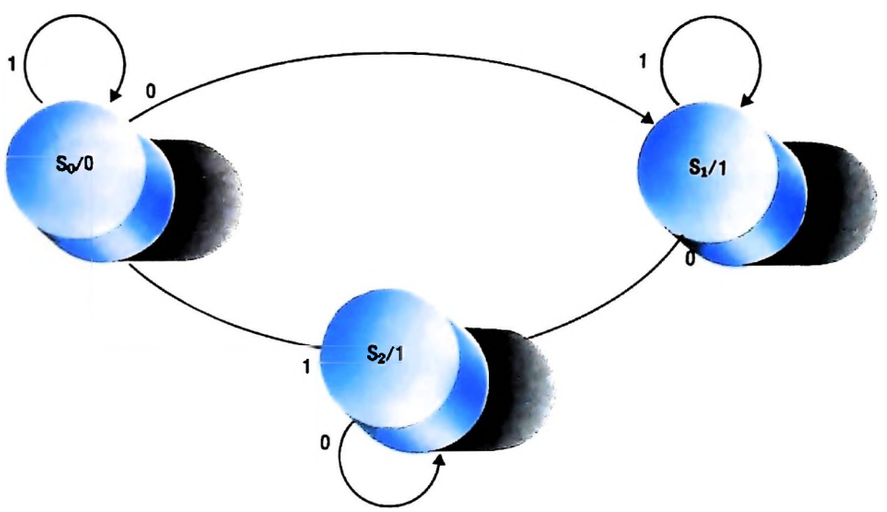
le dell'attività di questa macchina a stati finiti, nel momento in cui riceve in ingresso la successione di caratteri 01011 (nell'ordine da sinistra verso destra).

I grafi di transizione

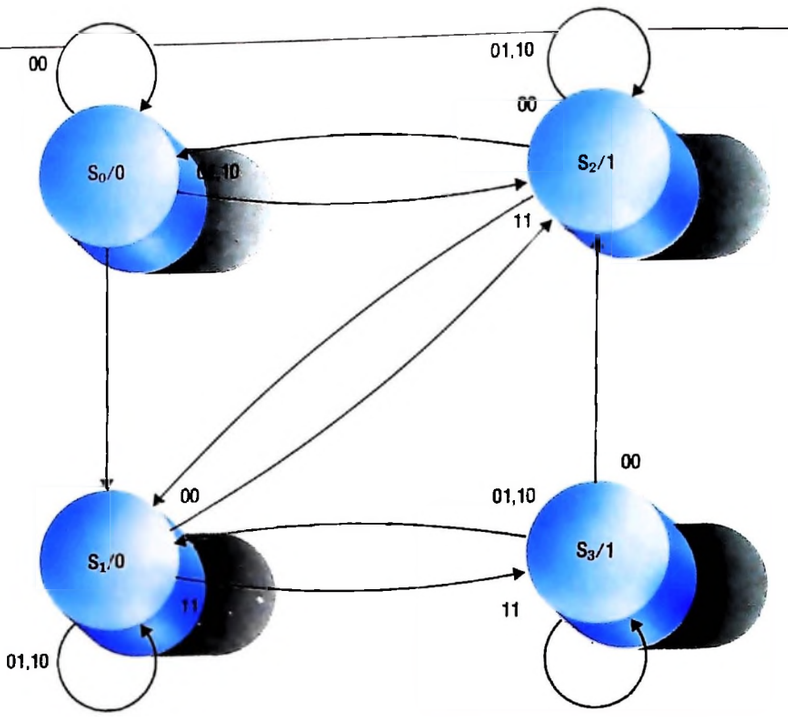
Il funzionamento di una macchina a stati finiti può essere rappresentato anche in un'altra forma grafica, che ha il pregio di essere molto chiara e intuitiva: mediante i *grafi di transizione* (o *diagrammi di transizione di stato*). I grafi di transizione sono figure costituite da vertici (raffigurati normalmente come cerchi, ma a volte anche come rettangoli, per comodità), chiamati *nodi*, collegati fra loro da segmenti orientati (raffigurati normalmente come archi con una freccia). Ogni nodo rappresenta uno stato ed è etichettato con il nome dello stato e il relativo output; da ogni nodo si dipartono tanti archi quanti sono i possibili input, e ciascuno punta al nodo che rappresenta lo stato successivo (funzione di quel- l'input e dello stato rappresentato dal nodo di partenza). Per convenzione, quando da un nodo si dipartono più archi che puntano allo stesso nodo successivo, per input diversi, se ne traccia uno solo e lo si etichetta con tutti gli input relativi.
In effetti, è più difficile descrivere a parole un grafo di transizione che seguirne l'andamento in un disegno: la figura in basso presenta il grafo di transizione corrispondente alla macchina *M* di cui abbiamo già visto la rappresentazione in forma di tabella.

Un automa che somma numeri binari

Come si lega il concetto di macchina a stati finiti (o *automa finito*, come viene anche detto) con il calcolatore? Un esempio di automa finito che esegue la somma di due numeri binari può darne un'idea molto meglio di qualunque dichiarazione di principio.
Perché un automa finito possa sommare due numeri binari,



Grafo di transizione per la macchina a stati finiti M. Di essa si è già vista la rappresentazione in forma di tabella.



Automa finito per la somma di numeri binari. È rappresentato, come si può vedere, sotto forma di grafo di transizione. Nella tabella riportata sotto vediamo come opera la macchina sommando i numeri 0011 e 0101. Ricordiamo che la condizione perché un automa finito possa sommare due numeri binari è che possa accettare in ingresso coppie di cifre binarie (00, 01, 10, 11).

Tempo	t_0	t_1	t_2	t_3	t_4
Input	11	10	01	00	—
Stato	S_0	S_1	S_1	S_1	S_2
Output	0	0	0	0	1

Numeri in ingresso: 0011 e 0101
Numero in uscita: 1000

deve poter accettare in ingresso coppie di cifre binarie (00, 01, 10, 11): le cifre corrispondenti dei due numeri da sommare. Per quanto riguarda l'output, dobbiamo ricordare i casi fondamentali della somma per i numeri binari, e cioè che 0 + 0 = 0, 0 + 1 = 1 + 0 = 1, 1 + 1 = 10 (ovvero 0 con riporto di 1). Sommando numeri di più cifre, può presentarsi anche il caso di 1 + 1 + 1, che dà 1 con riporto di 1.
I casi che si possono presentare possono essere ridotti a 4, in funzione dell'output: la somma delle due cifre immesse, cioè, può dare come risultato parziale 0, 0 con riporto; 1, 1 con riporto. I quattro casi corrisponderanno a quattro stati della macchina, S_0, S_1, S_2, S_3; S_0 è anche lo stato iniziale in cui si trova la macchina. Per ogni stato dobbiamo tener conto analogamente, come abbiamo visto, di quattro casi possibili (le quattro combinazioni di cifre che possono venire immesse); gli archi che si dipartono da ciascun nodo del grafo di transizione, tuttavia, saranno in tutti i casi solo tre, perché i casi 01 e 10 sono sempre identici (è la proprietà commutativa dell'addizione: l'ordine dei numeri nella somma è indifferente per il risultato).

L'automa finito per la somma di numeri binari è rappresentato nella figura in alto sotto forma di grafo di transizione. Vediamo con un esempio come funzioni la macchina: sommiamo i numeri 0011 e 0101. Per effettuare la somma, dobbiamo sempre partire dai bit meno significativi, cioè quelli più a destra: la successione delle coppie di cifre binarie in ingresso sarà perciò 11, 10, 01. All'istante t_0 l'automa è nello stato iniziale S_0 e riceve in ingresso la coppia 11; a t_1 passerà di conseguenza nello stato S_1, stamperà di conseguenza uno 0 (è il caso del riporto) e riceverà in ingresso la seconda coppia, 10. A t_2 rimane di conseguenza nello stato S_1: stampa nuovamente uno 0 (avevamo un 1 di riporto) e riceve in ingresso la terza coppia: 01. A t_3, ancora una volta, lo stato è immutato: viene stampato ancora uno 0 e viene accettata in ingresso la coppia 00. A t_4, infine, la macchina è passata nello stato S_2 e stampa un 1.
A questo punto non ci sono più dati in ingresso ed è come se la macchina continuasse d'ora in poi a ricevere coppie di 0: a t_5 si troverà in S_0 e continuerà a rimanervi indefinitamente, fino all'ingresso di nuovi dati significativi.

CONTROLLO (I)

Vediamo la realizzazione pratica di alcuni circuiti di controllo.

Nella Uamicro I abbiamo visto come, trattandosi di un sistema a pochi registri, il controllo può essere realizzato con circuiti combinatori. Nella Uamicro II invece, a seguito della maggiore quantità di registri da controllare, una forma più efficiente di controllo risulta essere la tecnica delle "parole fisse di controllo" attivate opportunamente ogni volta che è necessario. Nella Uamicro III la quantità di registri è alta e, se vogliamo aumentare ancora di più le prestazioni, è suscettibile di aumento, per cui l'unica via possibile per la creazione di un controllo efficiente è quella usata nella Uamicro II, naturalmente cercando di perfezionarla.
La tecnica delle "parole fisse", comunemente chiamata anche "tecnica ROM", si basa sull'uso delle memorie ROM che sono, come già sappiamo, memorie di sola lettura. Sfruttando questa caratteristica possiamo immagazzinare tutte le "parole" necessarie per eseguire le istruzioni presenti nel microprocessore per poi attivarle una dopo l'altra in forma opportuna.
Prima di vedere la realizzazione pratica di alcuni circuiti di controllo, definiamo alcuni concetti fondamentali come:
— ISTRUZIONI PRIMARIE
— MACROISTRUZIONI O ISTRUZIONI COMPOSTE
— MICROROUTINE

Istruzioni primarie

Le istruzioni primarie sono le istruzioni ad azione singola come LDA, LDB ecc. La loro quantità e funzione dipende dall'uso a cui è preposto il microprocessore.

Macroistruzioni o istruzioni composte

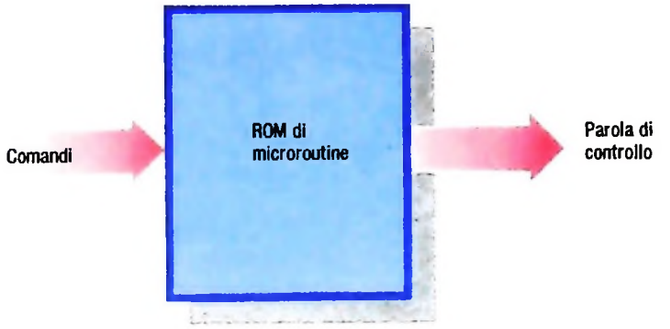
S'intende un insieme di istruzioni primarie tipo LDA, STA ecc. che insieme eseguono operazioni di ordine superiore di complessità; sono necessarie per migliorare la velocità dei microprocessori e per cercare di avvicinarsi il più possibile alle esigenze del software.

Microroutine

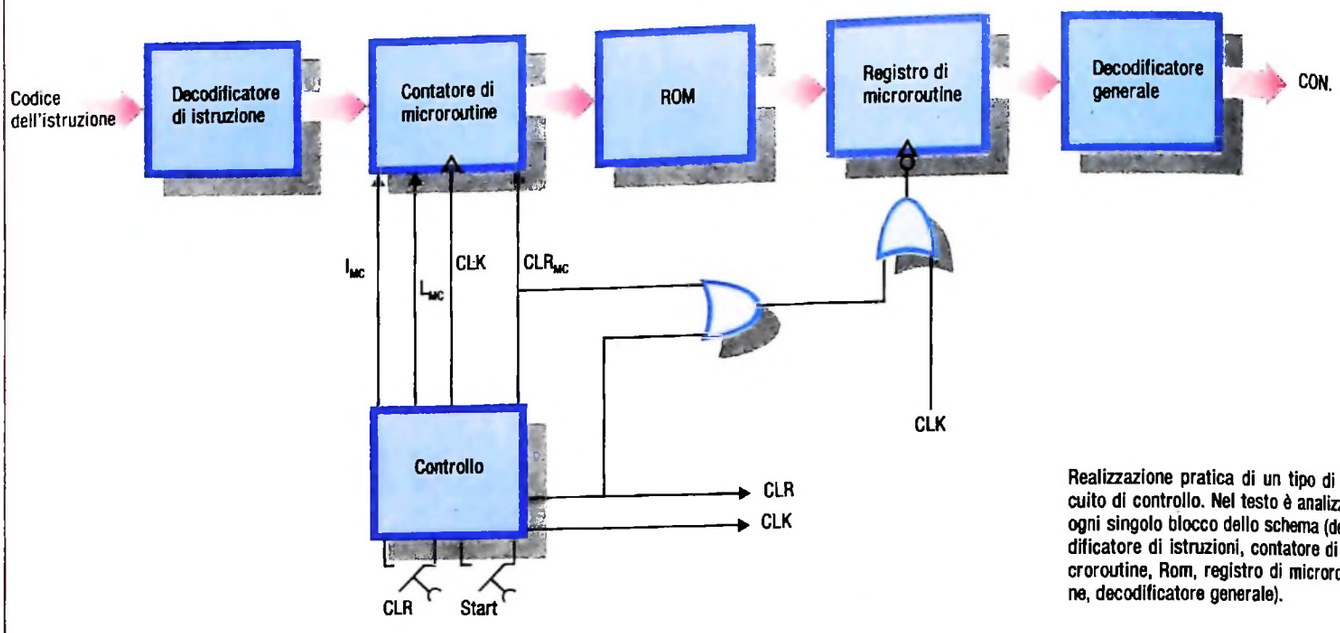
Sono le sequenze di azioni che il microprocessore compie per "interpretare" ed "eseguire" un'istruzione (abbiamo già visto questo concetto parlando delle fasi di FETCH ed EXECUTE di un'istruzione). Esse sono esclusive di ogni microprocessore perché dipendono dai registri presenti in esso.

Architettura di controllo

In questa versione prenderemo in considerazione due soluzioni di questo problema. Naturalmente queste non sono le uniche possibili in quanto la necessità di velocità, principalmente, e la necessità di ridurre gli spazi, fa sì che vi sia una ricerca continua di soluzioni nuove e più efficienti. In tutti i modi una soluzione che sembra finora insuperata è basata sull'uso della ROM (figura in basso), opportunamente programmata, che a ogni comando o indirizzo di entrata emette una "parola di controllo". Su questo tipo di soluzione, ci baseremo per risolvere il nostro problema.



Uso della memoria Rom per il problema dell'apertura di controllo. Ad ogni indirizzo di entrata, la Rom emette una "parola di controllo".

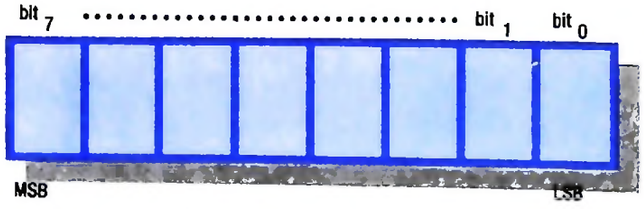


Realizzazione pratica di un tipo di circuito di controllo. Nel testo è analizzato ogni singolo blocco dello schema (decodificatore di istruzioni, contatore di microroutine, Rom, registro di microroutine, decodificatore generale).

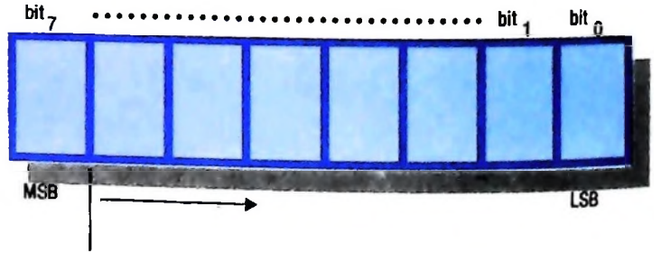
Prendiamo la figura sopra e partendo da sinistra verso destra, come di consueto, analizziamo ogni blocco dandone l'opportuna spiegazione.

Decodificatore di istruzioni

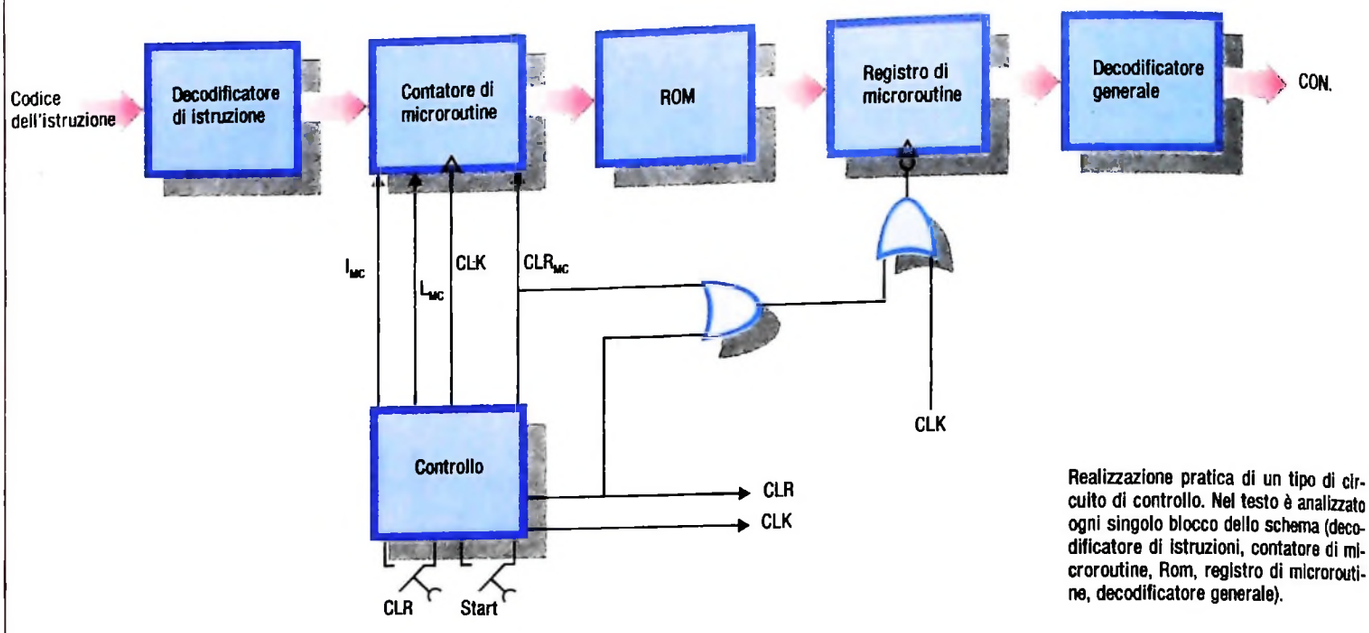
Nella Uamicro III, come in tutti i microprocessori a 8 bit, abbiamo a disposizione ben 256 ($2^8 = 256$) codici diversi da assegnare alle istruzioni. La tecnica di assegnazione del codice all'istruzione varia, naturalmente, da fabbricante a fabbricante, però ci sono alcuni principi che sono comuni a tutti. Prima di tutto si suddivide il blocco di istruzioni in due gruppi, oppure, a volte, in tre, comprendenti rispettivamente le istruzioni MRI, le istruzioni di tipo OPERAZIONE e le istruzioni di 1/0 (entrata e uscita). Le prime, come sappiamo, sono quelle che per completarsi hanno bisogno di ritornare alla memoria per prendere gli operandi, mentre gli operandi delle seconde sono presenti nei registri interni del microprocessore. Le ultime sono le istruzioni dedicate alle operazioni di entrata e uscita dati. Questa prima suddivisione viene fatta, quasi sempre, attraverso i bit di più alto valore e nel nostro caso specifico, dal momento che abbiamo solo le istruzioni MRI e OPERAZIONE, con l'MSB (figura a sinistra in basso), che ha il valore zero quando l'istruzione è MRI, mentre ha il valore uno quando l'istruzione è di tipo OPERAZIONE. Per ognuno di questi due blocchi esiste un'altra suddivisione che riguarda, per le MRI, il modo di indirizza-



MSB = 0 → istruzione MRI
MSB = 1 → istruzione OPERAZIONE



Campo di indirizzo

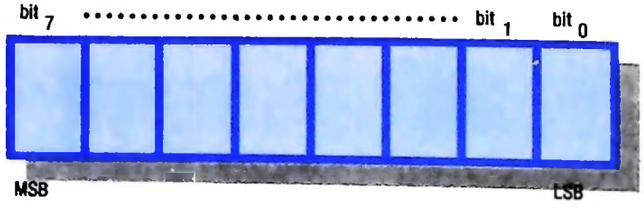


Realizzazione pratica di un tipo di circuito di controllo. Nel testo è analizzato ogni singolo blocco dello schema (decodificatore di istruzioni, contatore di microroutine, Rom, registro di microroutine, decodificatore generale).

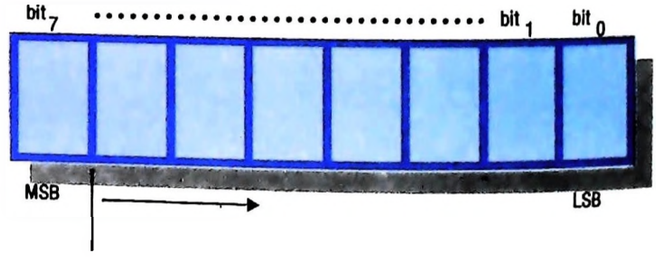
Prendiamo la figura sopra e partendo da sinistra verso destra, come di consueto, analizziamo ogni blocco dandone l'opportuna spiegazione.

Decodificatore di istruzioni

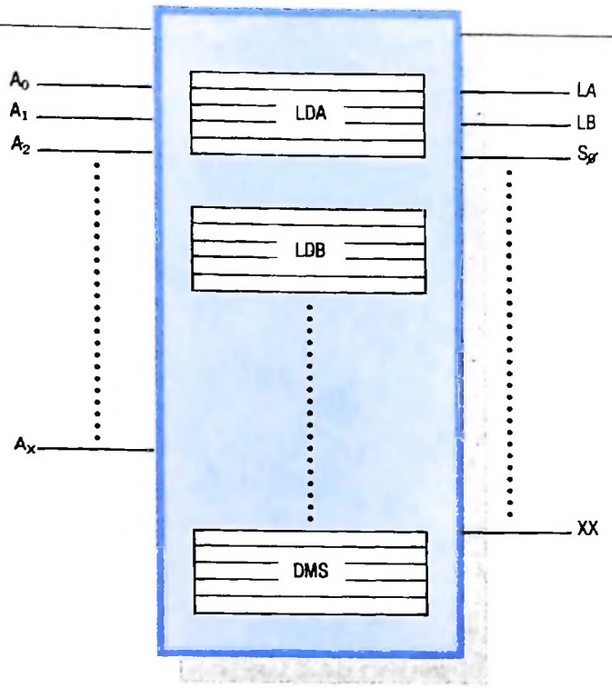
Nella Uamicro III, come in tutti i microprocessori a 8 bit, abbiamo a disposizione ben 256 ($2^8 = 256$) codici diversi da assegnare alle istruzioni. La tecnica di assegnazione del codice all'istruzione varia, naturalmente, da fabbricante a fabbricante, però ci sono alcuni principi che sono comuni a tutti. Prima di tutto si suddivide il blocco di istruzioni in due gruppi, oppure, a volte, in tre, comprendenti rispettivamente le istruzioni MRI, le istruzioni di tipo OPERAZIONE e le istruzioni di 1/0 (entrata e uscita). Le prime, come sappiamo, sono quelle che per completarsi hanno bisogno di ritornare alla memoria per prendere gli operandi, mentre gli operandi delle seconde sono presenti nei registri interni del microprocessore. Le ultime sono le istruzioni dedicate alle operazioni di entrata e uscita dati. Questa prima suddivisione viene fatta, quasi sempre, attraverso i bit di più alto valore e nel nostro caso specifico, dal momento che abbiamo solo le istruzioni MRI e OPERAZIONE, con l'MSB (figura a sinistra in basso), che ha il valore zero quando l'istruzione è MRI, mentre ha il valore uno quando l'istruzione è di tipo OPERAZIONE. Per ognuno di questi due blocchi esiste un'altra suddivisione che riguarda, per le MRI, il modo di indirizza-

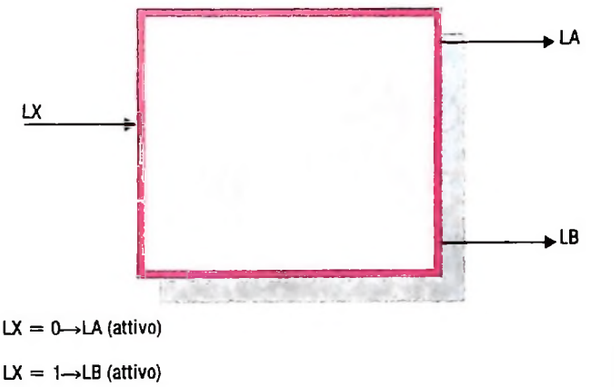


MSB = 0 → istruzione MRI
MSB = 1 → istruzione OPERAZIONE



Campo di indirizzo





Nella Rom vengono immagazzinate le parole di controllo per ogni istruzione. Da una parte viene introdotto il codice dell'istruzione, dall'altra viene estratta la parola di controllo. A destra: mediante l'uso di un decodificatore in uscita si può ridurre la parola che viene fuori dalla Rom. Invece di usare due linee, ne usiamo una sola.

mento che, oltre a permettere una riduzione del tempo di esecuzione delle istruzioni, rende più efficiente la stessa istruzione riuscendo a soddisfare tutta una gamma di situazioni particolari di software. Per le istruzioni di tipo OPERAZIONE, invece, si possono avere varianti nell'uso dei registri interni a volte presi singolarmente, altre a coppie o quadruple secondo le necessità e il fabbricante.
Per poter determinare tutte queste varianti si usano i bit che seguono l'MSB (figura a destra, pagina accanto) in quantità dipendente dal tipo, mentre gli ultimi bit che restano individuano l'istruzione. Se i bit non bastano, si ricorre all'uso di un altro byte. In questo caso, le istruzioni sono a più byte.

Contatore di microroutine

Questo registro immagazzina internamente con L_{MC} = 1 e al CLK l'indirizzo emesso dal registro precedente e lo passa immediatamente alla ROM. Nella fase successiva il controllo emette I_{MC} = 1 per cui, al CLK, l'indirizzo immagazzinato viene incrementato di uno e punta alla località che segue della ROM, dove è immagazzinata la "parola di controllo" della fase successiva e così via fino a quando il controllo non emette il comando CLR_{MC}, che cancella il registro interno.

ROM

In questa ROM (figura in alto a sinistra) sono immagazzinate le parole di controllo per ogni istruzione. Da una parte (sinistra) viene introdotto il codice dell'istruzione sotto forma di indirizzo, dall'altra (destra) viene estratta la parola di controllo con le sue linee di comando attivate o meno.

Registro di microroutine

Questo registro è un Buffer e come tale funziona: la parola di controllo viene immagazzinata alla transizione negativa del CLOK e resta fissa all'entrata del registro Decodificatore Generale.

Decodificatore generale

Quando siamo in presenza di un microprocessore del tipo Uamicro III o più grande, la quantità di registri interna è piuttosto elevata e di conseguenza anche la quantità della parola di controllo.
Se questa parola dovesse venir fuori tutta dalla ROM ci sarebbero dei problemi di velocità di trasmissione dati, e quindi, per ridurre questo problema, si ricorre all'uso di un altro decodificatore in uscita.
Per comprendere il funzionamento di questo decodificatore prendiamo per esempio i comandi di LOAD degli accumulatori A e B, cioè i comandi LA e LB: invece di usare due linee ne usiamo una sola che chiamiamo LX (figura in alto) e questa entra in un decodificatore che attiva la linea LB o LA secondo che il valore di LX è zero oppure uno.
Questo principio può essere esteso anche ad altri comandi e quindi la parola che viene fuori dalla ROM è molto più ridotta e quindi più facile da maneggiare.

Creazioni d'arte con le immagini elettroniche.

Di particolare interesse sono le immagini, come quelle riportate in queste pagine, che utilizzano tecniche miste. Molto spesso infatti, anche per ragioni economiche, i grandi studi di produzione televisiva uniscono alle immagini elettroniche sistemi di animazione tradizionale oppure filmati 'dal vero'.
In questo caso le immagini sono state digitalizzate con una tavoletta elettronica e quindi stampate in bianco e nero.
Successivamente le stampate sono state colorate con acquarello. Possiamo ancora osservare che la stampante consente, fra l'altro, la deformazione dei rapporti tra i lati dell'immagine. Questa possibilità della stampante è evidenziata nella seconda immagine.

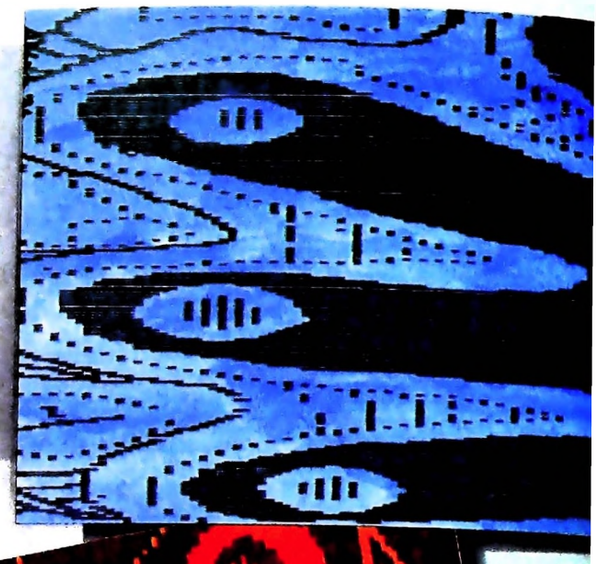


FOTO DI ADRIANO ABBADO

Lezione 43

Il microplotter: spostamenti dei pennini

Nella lezione precedente avevamo costruito un programma che permetteva di visualizzare sul microplotter un messaggio, con differenti dimensioni dei caratteri; in quell'occasione, il microplotter stampava i vari messaggi consecutivamente, senza andare a capo dopo ciascuno; avevamo visto quindi la necessità di disporre di comandi per spostare i pennini, in modo da cambiare la riga di stampa.
Purtroppo, il programma stampava il messaggio con dimensioni crescenti, quindi, per poter andare adeguatamente "a capo", sarà necessario che l'interlinea effettuata sia di dimensioni proporzionali al carattere da stampare, in modo da evitare che le varie righe si sovrappongano.
Per andare a capo useremo il comando R, brevemente introdotto la volta precedente, che permette SPOSTAMENTI RELATIVI rispetto all'attuale posizione del pennino; tale comando ha la seguente forma:

```
RX,Y
```

ove

- X, indica il numero di colonne di cui il pennino deve avanzare (se il valore è positivo) o retrocedere (se è negativo);
- Y, indica il numero di righe di cui deve risalire (se il valore è positivo) o scendere (se è negativo).

Come al solito, il comando sarà inviato mediante l'istruzione LPRINT.
Poiché il nostro programma stampa caratteri di dimensione crescente, è necessario che ogni interlinea sia di dimensione adeguata per contenere il carattere; quindi sarà necessario ancora una volta comporre il comando R facendo dipendere lo spostamento dal valore della variabile I, che determina la dimensione del carattere:

- poiché dobbiamo tornare indietro sulla riga e scendere di varie righe, i due valori X e Y dovranno essere ambedue negativi
- poiché I varia da 0 a 63, dovremo spostarci indietro e sotto di un numero di posizioni che dipendono da I, e che potrebbero essere rispettivamente della forma:

```
X = - K * (I+1)
```
e
```
Y = - H * (I+1)
```

ove K e H sono opportuni valori da definire, e in modo tale che dopo avere stampato il messaggio con I=0 ci si sposti indietro di K posizioni e sotto di H (per questo motivo si è sommato 1 al valore di I).
Dopo aver effettuato qualche tentativo, vedremo che due valori adeguati per le due costanti sono:

K=30 e H=10

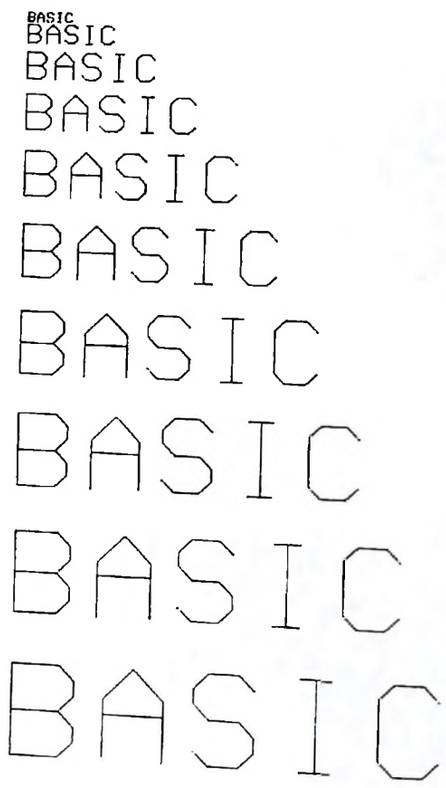
In più, potremo aggiungere altre 5 posizioni fisse di interlinea verticale, per aumentare leggermente in modo costante la distanza tra le scritte.
La costruzione del comando di posizionamento sarà quindi:

```
67 LET P$="R-"+STR$(30*(I+1))+",-"+STR$(10*(I+1)+5)
```

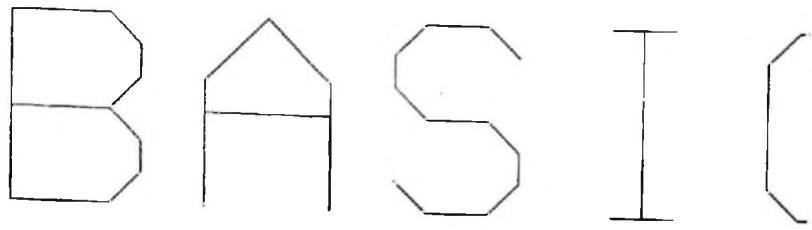
ove si sono concatenati i vari caratteri, compreso "—" e "," ai valori calcolati in dipendenza di I.
Il programma risulterà quindi:

```
10 LPRINT CHR$(18) 'Graphic mode
30 FOR I=0 TO 63
40 REM Costruisce il comando
50 LET S$="S"+STR$(I)
60 LPRINT S$
65 LPRINT "PBASIC"
67 LET P$="R-"+STR$(30*(I+1))+",-"+STR$(
10*(I+1)+5)
68 LPRINT P$
70 NEXT I
```

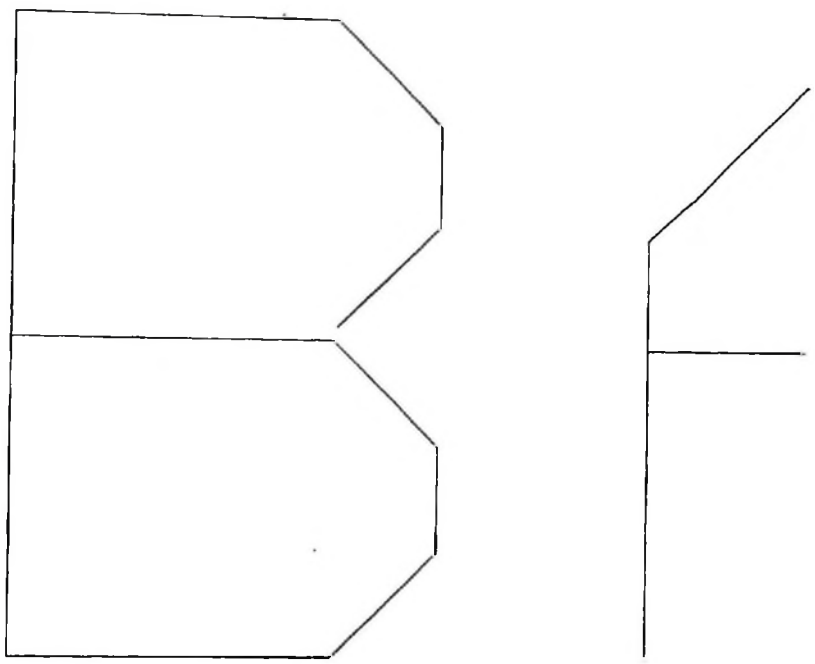
L'esecuzione del programma ci fornisce proprio il risultato desiderato: le prime 12 stampe risultano:



Dopo un certo numero di altre stampe, la dimensione dei caratteri fa uscire la scritta dal foglio:



e l'ultima stampa ci mostra una lettera B di dimensioni enormi:



Nella costruzione dei comandi di posizionamento ci siamo mossi un po' per tentativi: avremmo potuto fare per bene i conti delle dimensioni dei caratteri in funzione del valore di I, e quindi scegliere adeguatamente i valori delle costanti K e H; di fatto la procedura "per tentativi" è adeguata al nostro caso, e ci ha fornito una stampa che permette di avere la misura esatta di tutte le lettere al variare della dimensione, per poterne fare uso successivamente a fronte di necessità di nuovi posizionamenti.
Il modo con cui abbiamo operato è stato comunque permesso dal fatto che spostamenti sulla riga di valori troppo grandi fanno arrestare il pennino al margine sinistro in ogni caso.
Già nella lezione precedente abbiamo visto come è possibile calcolare la dimensione dei caratteri in funzione del comando S; avevamo visto infatti che un carattere elementare occupa un rettangolo di 4 punti di base e 6 di altezza, e ciò è ottenuto con

un comando S0; nel caso di un comando Sn (con n pari al più a 63) si ottengono rispettivamente

base = 4 * (1+n) punti
altezza = 6 * (1+n) punti

Da qui possiamo ricavare il numero di caratteri che può essere contenuto in una linea; infatti, sapendo che una riga contiene 480 punti, il numero di caratteri in essa stampabili in funzione di un comando Sn diviene:

$$\text{n° di caratteri su una linea} = \frac{480}{6 \times (1+n)}$$

che tiene conto del fatto che, oltre ai 4 * (1+n) punti occupati fisicamente dal carattere, sono presenti altri 2 * (1+n) punti di spaziature tra i vari caratteri.
Un ulteriore problema è costituito dalla spaziatura di interlinea: quanto spazio viene lasciato tra una riga e la successiva? Tale distanza è pari all'altezza del carattere che viene stampato. Così, se costruiamo un programma che scrive diverse righe con il comando S0, l'altezza del carattere è di 6 punti e tale sarà la dimensione dell'interlinea. Forse è meglio porre la questione in termini diversi: la distanza tra la base dei caratteri di una linea stampata e la base della successiva base è pari al *doppio* dell'altezza del carattere. Ciò rende conto del fatto che, con S0, lo spazio tra due righe è di 6 punti (in quanto la base della seconda riga alta 6 punti parte a 2 * 6 = 12 punti sotto la linea precedente), ma rende anche conto del fatto che, se dopo la stampa cambiano la dimensione del carattere, questo potrebbe essere troppo grande rispetto allo spazio lasciato, e sovrapporsi alla linea precedente. Da qui la necessità del comando R. Ora però conosciamo anche la regola, e potremmo fare con precisione i conti di "quanto spostarci".

Cosa abbiamo imparato

In questa lezione abbiamo imparato:

- il comando di microplotter R che permette di posizionare il pennino dello stesso con riferimento alla posizione in cui si trova.
- la dimensione della "base" dei caratteri in funzione del comando Sn (già visto precedentemente), pari a 4 * (1+n) punti;
- la dimensione dell' "altezza" dei caratteri in funzione del comando Sn, pari a 6 * (1+n) punti;
- la "capacità" di una riga, espressa in numero di caratteri stampabili in funzione del comando Sn, pari a 480/(6 * (1+n));
- lo spazio di interlinea lasciato tra la base di una riga e la base della successiva, pari a due volte l'altezza dell'ultima riga stampata.

STRUTTURE DI CONTROLLO E RICORSIONE IN LISP

Un linguaggio che, per le sue origini, risente molto di formalismi matematici.

Il nome del suo creatore, John McCarthy, che lo ideò intorno alla fine degli anni Cinquanta, è legato alla teoria del "lambda-calcolo", un formalismo matematico che è completamente riflesso dalla struttura del linguaggio.
In particolare il Lisp non presenta le strutture di controllo così come noi siamo abituati a pensarle negli abituali linguaggi di programmazione; osserviamo infatti come è possibile realizzare sequenze, selezioni e iterazioni con il Lisp.

Sequenza in LISP

Supponiamo di voler prendere l'ultimo elemento di una lista che rappresenta un'espressione aritmetica con due operandi e un operatore; avendo a disposizione solo gli operatori visti, dovremo:

- partire dalla lista iniziale, supponiamo (a+b)
- prelevarne il CDR, ottenendo (+b)
- prelevare il CDR del risultato, ottenendo (b)
- prelevare il CAR dell'ultimo risultato, ottenendo b.

Per realizzare tale sequenza di esecuzione, in Lisp, è necessario innestare tutte le operazioni nell'ordine inverso a quello dell'esecuzione, e quindi scrivere:

```
(CAR  (CDR  (CDR  '(a + b) ) ))
```

In sostanza, al posto di una struttura di controllo che permetta di effettuare le operazioni in sequenza, dobbiamo ricordare che un programma Lisp è una sola espressione, e che la sequenza è ottenuta come ordine di valutazione dei vari argomenti.

Selezione in LISP

Esiste nel linguaggio Lisp una struttura di selezione, già illustrata precedentemente, si tratta della funzione COND, che corrisponde, approssimativamente a un IF..THEN..ELSE a scelte multiple.

Iterazione in LISP

In Lisp non esistono strutture di iterazione: per realizzare ripetizioni di operazioni è necessario usare la ricorsione.
Si tratta in sostanza di definire funzioni che vengono usate nella definizione stessa, e che quindi "ricorrono" nella loro definizione. Facciamo un esempio.
Il fattoriale di un numero n, che viene indicato come n!, è definito come:

n*(n-1)*(n-2)*...*3*2*1

cosicché:

4! = 4*3*2*1 = 24

Una convenzione stabilisce che 0! vale 1.
Un programma per il calcolo del fattoriale potrebbe quindi avere la seguente struttura:

- leggi n
- poni i:=1
- facendo variare i da 1 a n
  - poni fatt:= fatt*i
- stampa fatt

ma potrebbe anche sfruttare la definizione ricorsiva:
fatt (n):

```
se n=0 allora fatt:=1
  altrimenti fatt:=n*fatt (n-1)
```

In sostanza si dice che, se n è maggiore di zero, si può risolvere il problema riducendone la complessità, riportandolo cioè al fattoriale di n-1; prima o poi si arriverà al fattoriale di 0, di cui viene data la soluzione.
Per esempio, i passi per calcolare in modo ricorsivo fatt(3) risulterebbero:

3*fatt(2)
3*2*fatt(1)
3*2*1*fatt(0)
3*2*1*1

da cui finalmente ottenere il risultato.
In Lisp questo è il modo con cui si realizzano iterazioni. Per esempio, supponiamo di avere due liste e di volerne verificare l'eguaglianza: poiché EQ vale solo su atomi, possiamo costruire la seguente funzione (non introduciamo qui, per il momento, il modo di definire funzioni in Lisp, ma presentiamo la struttura della funzione così come questa verrebbe definita):

definizione di EQUAL su due liste A e B:

```
(COND
(    (EQ (CAR A) (CAR B))    (EQUAL (CDR A) (CDR B)  )
(T  F)
```

che sarebbe come dire:

- se due liste A e B hanno il primo elemento uguale, allora l'uguaglianza globale può essere verificata applicando la funzione EQUAL a quello che resta delle liste se togliamo il primo elemento;
- altrimenti le due liste sono sicuramente diverse.

La funzione EQUAL è quindi usata durante la sua definizione stessa.
Supponiamo di voler calcolare:

```
(EQUAL '(A B C) '(A B D))
```

le operazioni fatte risultano:

- A uguale ad A, e quindi (EQUAL' (B C)' (B D))
- B uguale a B, e quindi (EQUAL' (C)' (D))
- C diverso da D, e quindi il valore è F.

Di fatto siamo riusciti a ottenere un'iterazione di confronti su un dato che continuava a ridurre la sua dimensione.
Numerose versioni di Lisp hanno "sporcato", se così si può dire, la pulizia formale del linguaggio introducendo vere e proprie strutture di controllo, facilitando spesso la vita al programmatore.
Dal momento però che il nostro scopo è quello di illustrare la filosofia del linguaggio, noi ci atterremo durante la nostra trattazione alle versioni "rigorose" in cui tali estensioni al linguaggio non sono disponibili.

Un esempio interessante

Sfruttando le funzioni CSET e DEFUN illustrate nei riquadri, costruiamo un insieme di operatori che esaminano frasi italiane.
Possiamo innanzitutto definire insiemi di "parti del discorso", come:

```
(CSET 'ARTICOLIMS '(IL LO) )
(CSET 'ARTICOLIFS '(LA) )
(CSET 'ARTICOLIMP '(I GLI) )
(CSET 'ARTICOLIFP '(LE ) )
```

che raccolgono gli articoli rispettivamente maschili e femminili, singolari e plurali; quindi:

```
(CSET 'NOMIMS '(CANE GATTO BUE MASCHIO) )
(CSET 'NOMIFS '(DONNA PECORA MUCCA PANNA) )
(CSET 'NOMIMP '(CANI GATTI BUOI MASCHI) )
(CSET 'NOMIFP '(DONNE PECORE MUCCHE PANNE) )
```

che raccolgono nomi maschili e femminili; quindi:

```
(CSET 'VERBIS    '(AMA MANGIA VEDE))
(CSET 'VERBIP    '(AMANO MANGIANO BEVONO)
```

per le terze persone singolari e plurali di verbi transitivi.
Possiamo ora definire una funzione che dice se un atomo è presente in un certo insieme:

```
(DEFUN 'PRESENTE (ELEM INSIEME)
     (COND
          ((EQ (CAR INSIEME) NIL) F)
          ((EQ (CAR INSIEME) ELEM) T)
          (T  (PRESENTE ELEM (CDR INSIEME))
```

La funzione si domanda se l'insieme in cui si cerca è vuoto; in caso affermativo, l'elemento è assente di sicuro; altrimenti verifica se è il primo elemento dell'insieme e, in caso affermativo, risponde che l'elemento è presente; altrimenti, riapplica la funzione stessa sull'insieme a cui ha tolto il primo elemento. È chiaro che, continuando a togliere elementi, si arriverà all'insieme vuoto, a meno di non incontrare prima l'elemento cercato.
Possiamo ora costruire primitive più "alte":

```
(DEFUN 'ARTICOLOMS (ELEM) (PRESENTE ELEM ARTICOLIMS) )
```

che verifica se un elemento è presente nell'insieme degli articoli maschili singolari; analoghe funzioni possono essere costruite per gli altri tipi di articoli, per i nomi e per i verbi; inoltre possiamo costruire un predicato che verifichi se una coppia di elementi è un soggetto o un complemento oggetto corretto, ovvero, se è una corretta "parte nominale" singolare o plurale:

```
(DEFUN 'PARTENOMINALES (X)
```

```
(COND    ( (ARTICOLOMS (CAR X)) (NOMEMS (CAR
         (CDR X)))   )
         ( (ARTICOLOFS (CAR X)) (NOMEFS (CAR
         (CDR X)))   )
         ( T  F)

(DEFUN 'PARTENOMINALEF (X)
     (COND
          ( (ARTICOLOMF (CAR X)) (NOMEFF (CAR
         (CDR X)))
         ( (ARTICOLOFF (CAR X)) (NOMEFF (CAR
         (CDR X)))
         ( T  F )
```

Ora possiamo costruire primitive che estraggono un presunto soggetto o un presunto verbo o un presunto complemento poiché abbiamo implicitamente supposto che ogni frase sia del tipo:

< soggetto > < verbo > < complemento >

ove soggetto e complemento sono della forma

< articolo > <nome >

potremo costruire le funzioni:

```
(DEFUN 'ESTRAISOGGETTO (X)
     (CONS (CAR X)
                    (CONS (CAR (CDR X)) NIL )    ))
```

che unisce insieme il primo elemento con la lista costituita dal secondo e da NIL.
Analogamente:

```
(DEFUN 'ESTRAIVERBO   (X)
          (CAR  (CDR  (CDR  X ) ) ) )
```

che, semplicemente, toglie i primi due elementi della lista ed estrae il terzo; ancora:

```
(DEFUN 'ESTRAICOMPLEMENTO   (X)
          (CDR (CDR (CDR X ) ) ) )
```

che toglie dalla lista i primi tre elementi.
A questo punto possiamo introdurre predicati del tipo:

```
(DEFUN 'SOGGETTOGIUSTO (X)
     (COND ((PARTENOMINALES (SOGGETTO X)) T)
           (T   (PARTENOMINALEF (SOGGETTO X))   ))
```

e analogamente predicati sul controllo del verbo e del complemento; ancora, possiamo verificare concordanze del tipo:

```
(DEFUN 'CONCORDA (X)
     (COND   ( (PARTENOMINALES (SOGGETTO X))
                                (VERBOS (VERBO X))  )
             ( (PARTENOMINALEF (SOGGETTO X))
                                (VERBOF (VERBO X))  )
```

A questo punto possiamo usare l'insieme di predicati che ci siamo messi a disposizione, per scrivere espressioni del tipo:

```
(CONCORDA '(IL CANE MANGIA LA MUCCA))
```

la cui risposta è affermativa, ovvero:

```
(ESTRAICOMPLEMENTO '(IL CANE MANGIA LA MUCCA))
```

il cui valore è

```
(LA MUCCA)
```

È facile ora intuire come, attraverso una serie più ricca di funzioni, sia possibile costruire programmi che elaborano simbolicamente informazioni, interpretando il linguaggio, analizzando strutture e così via.

CSET

Un altro operatore rilevante, presente in moltissime versioni di Lisp è CSET, che permette di assegnare un nome a un determinato valore; così l'espressione: (CSET 'ESPR '(A + B))
assume valore (A + B), ma ha un effetto collaterale: da questo momento in poi il nome ESPR verrà valutato come (A + B).
Così: (CAR ESPR) vale A
Si noti, invece, che (CAR 'ESPR) non è un'operazione legale, in quanto 'apice che precede il nome ESPR ne inibisce la valutazione, e in questo modo ESPR viene considerato un simbolo atomico, di cui non è possibile prelevare il CAR.

La definizione di funzioni in Lisp

Il programmatore Lisp ha la possibilità di definire proprie funzioni con un particolare operatore, che di solito può essere simile allo CSET, e assume il nome di DEFINE o di DEFUN. Tra le varie versioni di LISP, scegliamo qui quella che usa DEFUN.
La scrittura (DEFUN 'F (X) (CAR (CDR X)))
definisce una funzione di nome F che, applicata a un argomento X, ne estrae il CAR del CDR.
Così: (CSET 'A '(P + Q))
(F A)
calcola la funzione F applicata all'argomento A, estraendone prima il CDR, che vale (+ Q), e quindi il CAR, che vale "+".
Una funzione può avere più argomenti.

DATA-BASE RELAZIONALI

Come organizzare le relazioni e quali operazioni effettuare per manipolare un insieme di dati.

Nel precedente articolo abbiamo esaminato, senza scendere nei dettagli, quali possono essere i problemi dell'organizzazione dell'informazione e come si sia tentato di risolvere questi problemi con i vari modelli di data-base.
In questo articolo soffermeremo la nostra attenzione sul modello relazionale di data-base, che in questi ultimi anni ha riscosso un notevole successo.
Inizialmente vedremo in che modo devono essere organizzate le relazioni perché sia possibile operare in modo coerente sulla base di dati; quindi descriveremo le operazioni che normalmente si possono effettuare con un data-base relazionale per manipolare insiemi di dati.
La semplice definizione di relazione che abbiamo dato nel precedente articolo permette di definire un modello di data-base relazionale. Questo modello non possiede però le caratteristiche necessarie per poter utilizzare potenti strumenti matematici quali il calcolo dei predicati e l'algebra delle relazioni (o relazionale). Tali strumenti sono la base su cui poggiano i linguaggi di interrogazione di tipo pseudo-naturale, cioè quei linguaggi con sintassi e semantica simili a quelli del linguaggio naturale, mediante i quali è possibile interagire con il data-base per "estrarre" informazioni o modificare i dati già esistenti. Per esempio in uno di questi linguaggi è possibile, avendo definito una relazione impiegati con attributi nome, salario ed età, fare domande del tipo:

```
RETRIEVE (impiegati.nome)
WHERE
    impiegati.salario > 1 500 000
AND
    impiegati.età <25
```

per ottenere una lista dei nomi degli impiegati il cui salario è maggiore di 1 500 000 e la cui età è minore di 25 anni. Il significato delle parole retrieve, where e and, che sono parole riservate del linguaggio, è traducibile in italiano in "recupera", "dove" e "e".
Per poter applicare al modello relazionale il calcolo dei predicati e l'algebra relazionale, il modello dei dati deve essere trattato in termini insiemistici ed è necessario che la struttura delle relazioni soddisfi certi requisiti. Il procedimento di trasformazione che porta una relazione ad avere i requisiti richiesti viene detto "normalizzazione". Questo procedimento avviene, in generale, in tre passi successivi e le caratteristiche che la relazione viene ad avere a ogni stadio della normalizzazione sono dette prima, seconda e terza "forma normale".
L'importanza delle forme normali è legata ai vantaggi che esse offrono relativamente alla manutenzione e all'aggiornamento dei dati. Tramite le tre normalizzazioni che abbiamo descritto, un originario modello di dati viene trasformato in un modello caratterizzato da una coerenza logica che permette di effettuare modifiche e variazioni mantenendo una elevata integrità dell'insieme.
Sebbene i data-base relazionali non obblighino l'utente a scoprire tutte le possibili dipendenze funzionali tra i dati, richiedendo relazioni in terza forma normale, è importante ricordare che un modello normalizzato è una solida base per poter un domani effettuare modifiche strutturali del modello senza incorrere in gravi rischi di perdita dell'integrità.
Viceversa, in un modello concepito senza tenere conto delle normalizzazioni, future modifiche strutturali delle relazioni possono portare a inconsistenze logiche dovute alla non considerazione di legami di dipendenza funzionale a prima vista non appariscenti.

Prima forma normale

La prima forma normale tratta l'aspetto delle ennuple di una relazione. Una prima caratteristica che viene richiesta per un data-base relazionale è che i suoi attributi non siano decomponibili, cioè non siano a loro volta una relazione.
Una costruzione non ammessa è, pertanto, quella che prevede multipli elementi connessi ad un singolo attributo, così come avviene per l'attributo "figli" nel caso seguente:

impiegato			
nome	età	figli	coniuge
Rossi	50	Maria Paolo	Barbara

Detto in termini banali si richiede che una ennupla sia costituita da una sola riga nella tabella che rappresenta la relazione, senza che vi siano "sovrascritture" all'interno di alcun campo. La soluzione adottata per ridurre in prima forma normale una relazione come la precedente è normalmente basata sulla scissione della relazione originaria in due rela-

zioni, entrambe in prima forma normale.
Nel nostro caso la relazione "impiegato" verrà trasformata in una nuova relazione che chiameremo ancora "impiegato":

impiegato		
nome	età	coniuge
Rossi ...	50 ...	Barbara ...

alla quale è associata la relazione "figli":

figli	
padre	figlio
Rossi Rossi ...	Maria Paolo ...

Oltre a non avere elementi multipli in corrispondenza di alcun attributo di ciascuna ennupla, una relazione normalizzata deve soddisfare anche altre condizioni. In particolare deve esistere un sottoinsieme degli attributi tale che il contenuto dei campi relativi permetta di identificare univocamente una ennupla all'interno della relazione.
L'insieme di questi attributi viene detto chiave.
Ovviamente in una relazione non possono esistere due o più ennuple con la stessa chiave.

Seconda forma normale

La seconda forma normale tratta le relazioni tra campi chiave e campi non-chiave (o dipendenti). Perché una relazione possa essere definita in seconda forma normale si richiede che non vi siano ridondanze di informazione tra campi chiave e campi dipendenti.
Consideriamo ora la seguente relazione utilizzabile per un possibile inventario:

inventario			
oggetto	fornitore	quantità	indirizzo-fornitore
matite gomme penne ... matite ...	Rossi Rossi Rossi ... Bianchi ...	100 27 13 ... 95 ...	Via Torino 23, Milano Via Torino 23, Milano Via Torino 23, Milano ... V.le Monza 122, Milano ...

La chiave in questo caso è costituita dai campi "oggetto" e "fornitore", ma il campo "indirizzo-fornitore" dipende direttamente solo dal campo "fornitore".
Ecco i principali problemi che si potrebbero avere con una struttura simile:

— l'indirizzo del fornitore viene ripetuto per ogni oggetto relativo allo stesso fornitore;

— se l'indirizzo del fornitore dovesse cambiare, si dovrebbero aggiornare tutte le ennuple relative;

— a causa della ridondanza, durante un aggiornamento si potrebbero avere dati inconsistenti, con ennuple indicanti indirizzi differenti per lo stesso fornitore;

— se in qualche momento non ci fosse in magazzino alcun oggetto di un certo fornitore, si perderebbe traccia dell'indirizzo di quel fornitore.

Per riportare in seconda forma normale una relazione come quella descritta dalla tabella precedente, dobbiamo decomporla nelle due relazioni:

inventario			
oggetto	fornitore	quantità	indirizzo-fornitore
matite gomme penne ... matite ...	Rossi Rossi Rossi ... Bianchi ...	100 27 13 ... 95 ...	Via Torino 23, Milano Via Torino 23, Milano Via Torino 23, Milano ... V.le Monza 122, Milano ...

e

fornitori	
nome	indirizzo
Rossi Bianchi ...	Via Torino 23, Milano V.le Monza 122, Milano ...

in cui la prima ha come chiave gli attributi "oggetto" e "fornitore" e la seconda il solo attributo "nome".

Terza forma normale

La terza forma normale viene violata quando esiste una dipendenza funzionale tra campi non-chiave. Anche in questo caso possiamo fare un esempio.
Nella relazione "colore parti auto":

colore
parti auto

modello	parte	colore	codice colore
131	cofano	rosso	100
131	portiera	rosso	100
131	parafango	rosso	100
...	...	...	...
127	cofano	bianco	110
127	portiera	blu	120
...	...	...	...

esiste chiaramente una ridondanza dovuta alla dipendenza funzionale tra il colore e il codice corrispondente, che sono tra loro in relazione biunivoca.
Anche in questo caso sarà necessario esprimere la relazione originaria tramite due relazioni prive di dipendenze funzionali tra le parti non-chiave:

colore
parti auto

modello	parte	colore
131	cofano	rosso
131	portiera	rosso
131	parafango	rosso
...	...	...
127	cofano	bianco
127	portiera	blu
...	...	...

colori e
codici

colore	codice
rosso	100
bianco	110
blu	120

In questa forma tutti gli attributi dipendenti sono legati solamente alla parte chiave della relazione, il che rende esplicite le dipendenze funzionali tra dati, che altrimenti non sarebbero emerse con chiarezza.

Operazioni su relazioni

Uno dei punti di forza di un data-base relazionale è la disponibilità di semplici ma potenti operatori relazionali.
Le basilari operazioni tra insiemi sono infatti applicabili alle relazioni.
È quindi possibile definire la "unione", la "intersezione" e la "differenza" tra relazioni.
Inoltre sono disponibili due altre operazioni di grande potenza: la "proiezione" e la "giunzione".
Le prime tre operazioni sono possibili tra relazioni con struttura analoga, cioè con colonne di attributi corrispondenti, mentre, come vedremo, le operazioni di proiezione e di join non richiedono questa caratteristica.

Unione

L'unione di due relazioni con analoga struttura combina le due relazioni e produce una relazione che contiene tutte le ennuple della prima relazione e della seconda, eliminando però le ripetizioni. Quindi, nel caso in cui ci siano nelle due relazioni iniziali due ennuple identiche, solo una di esse verrà riportata nella relazione prodotta.
Consideriamo il semplice esempio delle due relazioni "colore-auto" e "colore-interni":

colore auto

colore	codice-colore
rosso	100
bianco	110
blu	120

colore interni

colore	codice-colore
verde	130
nero	140
rosso	100
giallo	150
blu	120

l'operazione di unione tra le due relazioni produrrà come risultato una relazione "colori-totali":

colori-totali

colore	codice-colore
verde	130
nero	140
rosso	100
giallo	150
blu	120
bianco	110

in cui appare l'unione delle ennuple presenti nelle due relazioni originarie.
L'operazione di unione corrisponde quindi all'operatore logico "or" e permette di conoscere tutti gli elementi che appartengono a una certa categoria (relazione) "oppure" a un'altra.

Intersezione

La funzione di questa operazione è di generare una relazione in cui compaiono solo quelle ennuple che fanno parte di en-

trambe le relazioni originarie. L'intersezione può ridurre notevolmente l'insieme dei dati iniziali ed è un potente strumento di estrazione e selezione delle informazioni.
Applicando l'operatore intersezione alle relazioni "colore-auto" e "colore-interni", descritte sopra, si otterrebbe la seguente relazione:

colori comuni	
colore	codice-colore
rosso	100
blu	120

L'intersezione corrisponde all'operazione logica "and" e permette di estrarre dal data-base tutti quegli elementi che, appartenendo contemporaneamente a due categorie, hanno entrambe le caratteristiche espresse dalle due relazioni.

Differenza

La differenza tra due relazioni elimina dalla prima tutte quelle ennuple che esistono anche nella seconda relazione.
A differenza di quanto avviene per le operazioni di unione e intersezione, nella differenza l'ordine dei fattori è determinante per il risultato.
Effettuando la differenza tra le due relazioni "colore-auto" e "colore-interni", a seconda dell'ordine, si ottiene:

colore auto-interni	
colore	codice-colore
bianco	110

colore interni-auto	
colore	codice-colore
verde	130
nero	140
giallo	150

Proiezione

Molto spesso si può essere interessati solo a un sottoinsieme di una relazione, nel senso che non si desiderano tutti gli attributi, ma solo parte di essi. Ciò è equivalente ad avere una tabella con meno colonne di quella originale dalla quale si eliminano le ennuple identiche.
L'operazione di proiezione svolge proprio questo compito, generando una relazione ristretta rispetto a quella originaria.
Questa operazione dispone spesso di varie opzioni ed è arricchita da capacità di selezione condizionale.

Giunzione

Il ruolo della operazione di join è esattamente opposto a quello della proiezione. Permette di costruire una relazione espansa a partire da due relazioni originarie e, nella sua accezione più vasta in senso logico, questa operazione identifica il prodotto cartesiano tra due relazioni.
Il comportamento di un join viene controllato attraverso la comparazione dei valori assunti da alcuni specifici attributi.
L'operazione di join più diffusa è quella di equijoin, che richiede l'uguaglianza tra i valori assunti da alcuni attributi nelle due relazioni di partenza.
Effettuando tale operazione sull'attributo comune "parte" delle due relazioni "colore parti auto" e "tipo parti auto":

tipo parti auto		
parte	spessore	tipo
cofano	4 mm	laminato
cofano	6 mm	satinato
parafango	6 mm	laminato
portiera	4 mm	laminato
portiera	6 mm	satinato

si ottiene una relazione espansa "colore e tipo parti auto" in cui vengono combinate le ennuple delle due relazioni:

colore e tipo parti auto				
modello	parte	spessore	colore	tipo
131	cofano	4 mm	rosso	laminato
131	cofano	6 mm	rosso	satinato
131	portiera	4 mm	rosso	laminato
131	portiera	6 mm	rosso	satinato
131	parafango	6 mm	rosso	laminato
127	cofano	6 mm	bianco	laminato
127	cofano	4 mm	bianco	satinato
127	portiera	6 mm	blu	laminato
127	portiera	4 mm	blu	satinato

La join ha qui preso in considerazione la prima ennupla della prima relazione comparandola via via con tutte quelle della seconda.
Ogni volta che i due valori dell'attributo "parte" erano uguali si è creata una nuova ennupla nella relazione "colori e tipo parti auto". Questa operazione è stata quindi effettuata per ogni ennupla della prima relazione.
Naturalmente l'operazione di join può essere effettuata anche scegliendo criteri di comparazione tra attributi comuni diversi dall'uguaglianza.

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.

olivetti

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

NOME/COGNOME

VIA/N

CAP/CITTÀ

TELEFONO