

Dispense per la prima parte del corso di *Fondamenti di Architetture e Programmazione*

Paolo Boldi

Versione 1.02

Queste dispense sono distribuite dall'autore sotto la licenza Creative Commons 2.5^a. Tu sei libero:

- di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera
- di modificare quest'opera
- di usare quest'opera per fini commerciali

alle seguenti condizioni:

- *Attribuzione.* Devi attribuire la paternità dell'opera nei modi indicati dall'autore o da chi ti ha dato l'opera in licenza;
- ogni volta che usi o distribuisi quest'opera, devi farlo secondo i termini di questa licenza, che va comunicata con chiarezza;
- in ogni caso, puoi concordare col titolare dei diritti d'autore utilizzi di quest'opera non consentiti da questa licenza.

Le utilizzazioni consentite dalla legge sul diritto d'autore e gli altri diritti non sono in alcun modo limitati da quanto sopra.

^a<http://creativecommons.org/licenses/by/2.5/deed.it>

Indice

1	Introduzione	2
1.1	Notazioni	2
1.2	Dimostrazioni per induzione	3
2	Rappresentazione dei dati	5
2.1	Rappresentazione dei naturali	5
2.1.1	Sistemi di numerazione	5
2.1.2	Sistemi di numerazione posizionali	7

2.1.3	Sistemi di numerazione posizionali — proprietà	9
2.1.4	Conversioni da base decimale a base b	11
2.1.5	Conversioni rapide per le basi 2, 8, 16	14
2.1.6	Operazioni in base b	15
2.2	Computer e rappresentazione digitale delle informazioni	16
2.2.1	Un esempio: rappresentazione di immagini a toni di grigio	17
2.2.2	Rappresentazione dei naturali; riporto e overflow	18
2.3	Rappresentazione degli interi	20
2.3.1	Rappresentazione mediante modulo e segno	20
2.3.2	Rappresentazione in complemento a due	21
2.3.3	Operazioni in complemento a due e overflow	25
2.4	Rappresentazione di numeri razionali	28
2.4.1	Sistemi di numerazione posizionale generalizzati	28
2.4.2	Rappresentazione in virgola fissa (fixed-point)	30
2.4.3	Rappresentazione in virgola mobile (floating-point)	30
2.5	Rappresentazione dei caratteri	32
3	Elementi di algebra di Boole e circuiti	35
3.1	Funzioni booleane e tabelle di verità	36
3.2	Un esempio pratico	38
3.3	Funzioni di base ed espressioni logiche	40
3.4	Proprietà algebriche	43
3.5	Completezza	44
3.5.1	Alcune funzioni notevoli	46
3.6	Circuiti booleani	46
3.7	Realizzazione di un sommatore	47
3.7.1	Semisommatore a 3 bit	48
3.7.2	Il sommatore	50

1 Introduzione

1.1 Notazioni

In queste dispense useremo le seguenti notazioni:

- $x \in X$ indica che x è un elemento dell'insieme X (si legge “ x appartiene a X ”); $x \notin X$ indica che x non appartiene a X .
- $X \subseteq Y$ indica che X è un sottoinsieme di Y , cioè che ogni elemento di X è anche un elemento di Y .
- $X \subset Y$ indica che X è un sottoinsieme proprio di Y , cioè che $X \subseteq Y$ e $X \neq Y$.
- $X \cup Y$ indica l'unione degli insiemi X e Y ; $X \cap Y$ indica la loro intersezione; $X \setminus Y$ indica la loro differenza, cioè l'insieme degli elementi che stanno in X ma non in Y .

- \mathbf{N} indicherà l'insieme dei *numeri naturali*, cioè degli interi non negativi; in altre parole, $\mathbf{N} = \{0, 1, 2, \dots\}$.
- \mathbf{Z} indicherà l'insieme degli *interi*; cioè $\mathbf{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$.
- \mathbf{Q} indicherà l'insieme dei *numeri razionali*, cioè dei numeri esprimibili come rapporto di due interi; in altre parole, $\mathbf{Q} = \{p/q \mid p, q \in \mathbf{Z}, q \neq 0\}$.
- \mathbf{R} indicherà l'insieme dei *numeri reali*.

Ovviamente $\mathbf{N} \subset \mathbf{Z} \subset \mathbf{Q} \subset \mathbf{R}$. Il fatto che $\mathbf{Q} \neq \mathbf{R}$, cioè che esistano numeri reali che non sono razionali, può essere facilmente provato come segue:

Teorema 1. $\sqrt{2} \in \mathbf{R} \setminus \mathbf{Q}$, quindi \mathbf{Q} è un sottoinsieme proprio di \mathbf{R} .

Dimostrazione. Supponiamo per assurdo che $\sqrt{2} = p/q$ per qualche coppia di interi $p, q \in \mathbf{Z}$ con $q \neq 0$. Possiamo, senza perdita di generalità, supporre che p e q siano *relativamente primi*, cioè che non esista alcun intero $k > 1$ che divida sia p che q (se ne esistesse uno, potremmo semplificare la frazione dividendo per k sia il numeratore che il denominatore). Segue che $q\sqrt{2} = p$ e quindi, elevando al quadrato entrambi i membri, $2q^2 = p^2$; quindi p^2 è divisibile per 2. Ma se un quadrato perfetto è divisibile per 2, deve anche essere divisibile per 4. Quindi q^2 deve essere divisibile per 2. Ma allora sia p che q sono divisibili per 2, contraddicendo l'ipotesi che siano relativamente primi. \square

1.2 Dimostrazioni per induzione

Considerate una proprietà $\mathcal{P}(x)$ che possa essere espressa per ciascun numero naturale x , ad esempio:

$$\mathcal{P}(x) = x \text{ è un numero primo.}$$

Naturalmente, per ciascun naturale x , la proprietà $\mathcal{P}(x)$ può essere vera o falsa; verificare se per uno specifico valore di x la proprietà $\mathcal{P}(x)$ vale oppure no significa stabilire se la proprietà è vera per quello specifico valore; se consideriamo la proprietà indicata sopra, ad esempio, $\mathcal{P}(17)$ è vera (poiché 17 è un numero primo) mentre $\mathcal{P}(18)$ è falsa (poiché 18 non è un numero primo).

In molte circostanze, risulta utile considerare proprietà che sono sempre vere, cioè che sono vere per ogni numero naturale $x \in \mathbf{N}$. Dimostrare che una certa proprietà ha questa caratteristica, cioè che vale per ogni numero naturale, non può essere fatto in modo "diretto", cioè mostrando che la proprietà vale per ogni singolo numero, visto che l'insieme \mathbf{N} è infinito! Una tecnica che risulta utile per dimostrare che una certa proprietà $\mathcal{P}(x)$ vale per ogni x consiste nell'utilizzo del cosiddetto *principio d'induzione*:

Principio d'induzione. Una proprietà $\mathcal{P}(x)$ è vera per ogni naturale $x \in \mathbf{N}$ se e solo se:

- (*caso base*) $\mathcal{P}(0)$ è vera
- (*passo induttivo*) assumendo che $\mathcal{P}(k)$ sia vera (*ipotesi d'induzione*), si riesce a dimostrare che $\mathcal{P}(k+1)$ è vera.

Per mostrare come si usa il principio di induzione, facciamo vedere come lo si può utilizzare per dimostrare la validità della formula di Gauss:

Teorema 2. Per ogni $x \in \mathbf{N}$,

$$0 + 1 + 2 + \cdots + (x - 1) + x = \frac{x(x + 1)}{2}.$$

Dimostrazione. In questo caso, la proprietà da provare vera è l'uguaglianza che compare nell'enunciato del teorema; dobbiamo dimostrare che vale per ogni $x \in \mathbf{N}$, e quindi procediamo per induzione.

Caso base. Nel caso $x = 0$, il lato sinistro dell'uguaglianza vale 0; il lato destro vale $0 \cdot (0 + 1)/2 = 0$.

Passo induttivo. Consideriamo la somma $0 + 1 + 2 + \cdots + (k - 1) + k + (k + 1)$; per ipotesi d'induzione, $0 + 1 + 2 + \cdots + (k - 1) + k = \frac{k(k+1)}{2}$. Quindi:

$$\begin{aligned} 0 + 1 + 2 + \cdots + (k - 1) + k + (k + 1) &= \frac{k(k + 1)}{2} + (k + 1) = \\ &= \frac{k(k + 1) + 2(k + 1)}{2} = \frac{(k + 1)(k + 2)}{2}. \end{aligned}$$

Ma questo è esattamente quanto volevamo dimostrare per il passo induttivo (quello ottenuto è il secondo membro dell'uguaglianza del teorema valutato per $x = k + 1$). \square

La formula del Teorema 2 è nota come *formula di Gauss* e la leggenda vuole che lo stesso Gauss l'abbia dimostrata quando era ancora bambino. La sua dimostrazione, in realtà, non usava l'induzione, ma la seguente tecnica:

Dimostrazione. (Dimostrazione alternativa) Chiamiamo S il valore (incognito) della somma:

$$S = 0 + 1 + 2 + \cdots + (x - 1) + x.$$

Ovviamente, essendo la somma commutativa, vale anche che

$$S = x + (x - 1) + \cdots + 2 + 1 + 0.$$

Se ora sommiamo membro a membro le due eguaglianze, possiamo notare che ciascuna delle coppie di addendi del lato destro ha somma x ($0 + x = x$, $1 + (x - 1) = x$ ecc.). Quindi

$$2S = \underbrace{x + x + \cdots + x + x + x}_{x+1 \text{ volte}},$$

e cioè $2S = x(x + 1)$ da cui $S = \frac{x(x+1)}{2}$. □

2 Rappresentazione dei dati

2.1 Rappresentazione dei naturali

2.1.1 Sistemi di numerazione

Un *sistema di numerazione* è un modo sistematico per rappresentare un certo insieme di numeri; in questa sezione, ci riferiremo principalmente ai sistemi di numerazione per i numeri naturali (e, ove non diversamente specificato, quando diremo “sistema di numerazione” intenderemo riferirci a un sistema di numerazione per i numeri naturali). La necessità di un sistema di numerazione è immediata conseguenza del fatto che l’insieme \mathbf{N} è infinito, e pertanto occorre trovare un modo uniforme per identificarne in modo univoco gli elementi.

Ogni sistema di numerazione definisce:

- un insieme N di *numerali*: un numerale è una rappresentazione simbolica di un numero;
- una funzione $v : N \rightarrow \mathbf{N}$ che associa ad ogni numerale un numero (il numero che quel numerale rappresenta).

È importante distinguere con cura il numerale (cioè, la rappresentazione simbolica) dal numero che il numerale rappresenta. Questo è particolarmente difficile quando si discute del sistema di numerazione decimale, quello che noi usiamo quotidianamente, in quanto il numerale è anche il modo con cui abitualmente rappresentiamo quel numero: tendiamo, cioè, a dimenticare che, ad esempio, 116 è solo una rappresentazione simbolica (costituita da una sequenza di tre caratteri: un 1 seguito da un 1 seguito da un 6) di un numero intero (il numero “centosedici”).

Numeri romani. Il discorso diventa più chiaro se pensiamo a un sistema di numerazione diverso da quello decimale. Consideriamo, ad esempio, il sistema di numerazione adottato nell'antica Roma. Stavolta, i numerali sono i numeri romani; ad esempio $XIII$, IV , $MCDIX$ sono tutti elementi di N , e $v(XIII) = 13$, $v(IV) = 4$, $v(MCDIX) = 1409$. Naturalmente, descrivere con cura l'insieme N e la funzione v è complicato: l'insieme N è infinito (i suoi elementi sono quelli che nel linguaggio quotidiano chiamiamo "numeri romani") e la funzione v indica come si determina a quale numero corrisponde ciascun numerale romano. Proviamo, come esercizio mentale, a dare una descrizione il più possibile completa del sistema di numerazione¹. Un numerale romano è una sequenza di elementi scelti dall'insieme di cifre $A = \{I, V, X, L, C, D, M\}$; ognuno degli elementi di A ha associato un numero naturale, mediante una funzione $a : A \rightarrow \mathbf{N}$ definita da $a(I) = 1$, $a(V) = 5$, $a(X) = 10$, $a(L) = 50$, $a(C) = 100$, $a(D) = 500$, $a(M) = 1000$. Normalmente per valutare a che numero corrisponde un certo numerale basta sommare i valori delle cifre da cui è composto, con un'eccezione: quando una cifra ne precede un'altra di valore maggiore, alla prima è attribuito il segno meno. Così, ad esempio, $v(XIII) = 10 + 1 + 1 + 1 = 13$, ma $v(IV) = -1 + 5 = 4$ (perché la cifra I vale 1 ed è seguita dalla cifra V che vale 5). Similmente $v(MCDIX) = 1000 - 100 + 500 - 1 + 10 = 1409$ (C precede D , e I precede X).

Quando si tratta di stabilire la validità di un determinato sistema di numerazione, ci si basa principalmente sui seguenti criteri:

- *completezza*: un sistema è completo se ogni numero (nel nostro caso: ogni numero naturale) ha almeno una rappresentazione; cioè, se la funzione $v : N \rightarrow \mathbf{N}$ è suriettiva; talvolta, si dice che un sistema di numerazione è *completo per l'insieme* A per intendere che è in grado di rappresentare tutti i numeri dell'insieme $A \subseteq \mathbf{N}$;
- *univocità*: un sistema è univoco se numerali diversi rappresentano sempre numeri diversi; cioè, se la funzione $v : N \rightarrow \mathbf{N}$ è iniettiva.

I numeri romani non sono un sistema né completo né univoco: il sistema non è completo perché il numero 0 non è rappresentabile²; non è univoco perché, ad esempio, il numero 9 si può rappresentare sia come $VIII$ che come IX (benché la seconda rappresentazione sia di gran lunga la più comune). Oltre a ciò, il sistema di numerazione romano è poco adatto a svolgere calcoli; in

¹In realtà, il sistema dei numeri romani presenta numerose varianti: quella che descriviamo qui è solo la più comune.

²In realtà, questo è l'unico numero naturale non rappresentabile, pertanto potremmo dire che il sistema è completo per l'insieme $\mathbf{N} \setminus \{0\}$.

effetti, non esiste nessun modo sistematico per effettuare somme o sottrazioni (per non parlare di moltiplicazioni e divisioni) manipolando direttamente i numerali romani.

2.1.2 Sistemi di numerazione posizionali

Molto più adatti a questo scopo sono i cosiddetti *sistemi di numerazione posizionali*, fra i quali è compreso il sistema di numerazione che noi adottiamo comunemente, il cosiddetto “sistema decimale” (o, più precisamente, sistema di numerazione posizionale in base 10). In un sistema posizionale, ogni numerale è costituito da una sequenza di simboli, detti *cifre*: il valore di ogni cifra dipende però non solo dalla cifra, ma anche dalla posizione in cui la cifra compare all’interno della sequenza (questo spiega il termine “posizionale”). Così, ad esempio, il numerale 453 rappresenta un numero diverso rispetto al numerale 354, o al numerale 345, benché tutti e tre siano costituiti dalle stesse cifre.

Base e cifre. I sistemi posizionali differiscono fra loro per un parametro, chiamato *base*: il sistema adottato universalmente oggi usa la base 10, ma è nostro scopo studiare i sistemi posizionali in generale, e perciò lavoreremo con una base $b > 1$ generica. Fissata la base b , si devono scegliere b simboli, detti *cifre*, che hanno i valori convenzionali $0, 1, \dots, b-1$. Nel seguito, confonderemo le b cifre con i valori ad essi attribuiti, e quindi diremo (impropriamente) che le cifre sono $0, 1, \dots, b-1$ (benché, più precisamente, dovremmo dire che le cifre sono b simboli *arbitrari* ai quali si attribuiscono questi valori).

Posizione. Come abbiamo detto, un numerale è una sequenza di cifre. Ogni cifra compare in una posizione, e le posizioni si considerano *da destra verso sinistra*: la posizione 0 è la posizione più a destra, la posizione 1 è quella che la precede immediatamente e così via. Ogni posizione ha associato un valore che è la potenza di b corrispondente: quindi la posizione 0 ha associato il valore b^0 , la posizione 1 ha associato il valore b^1 ecc.:

...	Pos. k	Pos. 3	Pos. 2	Pos. 1	Pos. 0
	b^k			b^3	b^2	b^1	b^0

Per calcolare il valore di un numerale bisogna moltiplicare il valore di ciascuna cifra per il valore della posizione in cui compare, e sommare i risultati.

Alcuni esempi: da base b a base 10.

1. quale valore rappresenta il numerale 12 in base 5?

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 5^1 & 5^0 \\ \hline \end{array}$$

Quindi $1 \cdot 5^1 + 2 \cdot 5^0 = 1 \cdot 5 + 2 \cdot 1 = 5 + 2 = 7$.

2. quale valore rappresenta il numerale 372 in base 8?

$$\begin{array}{|c|c|c|} \hline 3 & 7 & 2 \\ \hline 8^2 & 8^1 & 8^0 \\ \hline \end{array}$$

Quindi $3 \cdot 8^2 + 7 \cdot 8^1 + 2 \cdot 8^0 = 3 \cdot 64 + 7 \cdot 8 + 2 \cdot 1 = 192 + 56 + 2 = 250$.

3. quale valore rappresenta il numerale 100010 in base 2?

$$\begin{array}{|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 1 & 0 \\ \hline 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \hline \end{array}$$

Quindi $1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$. Notate che in questo caso (cioè, quando la base è 2) il calcolo è equivalente a *sommare semplicemente le potenze di due che corrispondono alle posizioni in cui compaiono gli 1*. Cioè, $2^5 + 2^1 = 32 + 2 = 34$.

4. quale valore rappresenta il numerale $A70D$ in base 16 (posto che la cifra A valga 10, la cifra B valga 11, ..., la cifra F valga 15)?

$$\begin{array}{|c|c|c|c|} \hline A & 7 & 0 & D \\ \hline 16^3 & 16^2 & 16^1 & 16^0 \\ \hline \end{array}$$

Quindi $10 \cdot 16^3 + 7 \cdot 16^2 + 0 \cdot 16^1 + 13 \cdot 16^0 = 10 \cdot 4096 + 7 \cdot 256 + 0 \cdot 16 + 13 \cdot 1 = 40960 + 1792 + 0 + 13 = 42765$.

Osservazioni. Alcune osservazioni importanti:

- come abbiamo detto, se la base b è minore o uguale a 10, usiamo come cifre i simboli abitualmente adottati (0 per rappresentare il valore zero, 1 per rappresentare il valore uno ecc.); per le basi b maggiori di 10, occorre trovare dei simboli per le cifre (di valore) maggiori di 9 (come abbiamo fatto nell'esempio della base 16);

- si parla di numerazione binaria, ternaria, ottale, esadecimale per indicare i sistemi di numerazione in base $b = 2$, $b = 3$, $b = 8$, $b = 16$ rispettivamente;
- nello svolgere i calcoli si possono semplicemente omettere nella somma gli addendi che corrispondono alle posizioni in cui compare la cifra 0; in particolare, per il sistema binario in cui le sole cifre sono 0 e 1, è sufficiente sommare le potenze che corrispondono alle posizioni in cui compare 1;
- talvolta, per evitare confusioni, quando si scrive un numerale in base b e si vuole rendere *esplicito* quale sia la base, si indica b a pedice; così, per esempio, 103_9 è il numerale 103 in base 9 (che corrisponde al numero 84), mentre 103_5 è il numerale 103 in base 5 (che corrisponde al numero 28); si omette di scrivere la base quando la base stessa è 10, oppure quando la base è chiara dal contesto.

2.1.3 Sistemi di numerazione posizionali — proprietà

Per iniziare lo studio delle proprietà dei sistemi di numerazione posizionali, limitiamoci inizialmente a considerare i numerali che si possono scrivere in base b usando k cifre. Poiché ogni cifra può essere scelta in b modi diversi, ci saranno in tutto b^k numerali di questo tipo. Ad esempio, se consideriamo la base 2 ($b = 2$) e assumiamo $k = 4$, i numerali possibili saranno $2^4 = 16$, e precisamente: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111.

Per discutere la situazione in modo più approfondito, conviene dimostrare il seguente teorema:

Teorema 3. Per ogni $b > 1$, e per ogni $k \in \mathbf{N}$

$$(b-1) \cdot b^k + (b-1) \cdot b^{k-1} + \dots + (b-1) \cdot b^2 + (b-1) \cdot b^1 + (b-1) \cdot b^0 = b^{k+1} - 1.$$

Dimostrazione. Per induzione su k .

Caso base. Per $k = 0$, il lato sinistro dell'uguaglianza vale $(b-1) \cdot b^0 = b-1$; il lato destro vale $b^1 - 1 = b-1$.

Passo induttivo. Vogliamo dimostrare che

$$(b-1) \cdot b^{k+1} + (b-1) \cdot b^k + \dots + (b-1) \cdot b^2 + (b-1) \cdot b^1 + (b-1) \cdot b^0 = b^{k+2} - 1,$$

usando l'ipotesi d'induzione

$$(b-1) \cdot b^k + (b-1) \cdot b^{k-1} + \dots + (b-1) \cdot b^2 + (b-1) \cdot b^1 + (b-1) \cdot b^0 = b^{k+1} - 1.$$

Il lato sinistro dell'uguaglianza da dimostrare si può scrivere, usando l'ipotesi d'induzione, come

$$(b - 1) \cdot b^{k+1} + b^{k+1} - 1$$

che è uguale a $(b - 1 + 1) \cdot b^{k+1} - 1 = b^{k+2} - 1$. \square

Grazie a questa proprietà, possiamo dimostrare che:

Teorema 4. *I numeri rappresentabili in base $b > 1$ mediante numerali di k cifre sono tutti e soli i valori compresi fra 0 e $b^k - 1$ (e ciascuno in modo univoco).*

Dimostrazione. (a) Il più grande valore rappresentabile è $b^k - 1$. Infatti:

$$\begin{array}{|c|c|c|c|c|c|} \hline b-1 & b-1 & \dots & \dots & b-1 & b-1 \\ \hline b^{k-1} & b^{k-2} & & & b^1 & b^0 \\ \hline \end{array},$$

vale $(b - 1) \cdot b^{k-1} + \dots (b - 1) \cdot b^0$. Usando il Teorema 3, questa somma vale $b^k - 1$.

(b) Due diversi numerali di k cifre rappresentano numeri diversi. Supponiamo che i numerali $c_{k-1}c_{k-2} \dots c_1c_0$ e $d_{k-1}d_{k-2} \dots d_1d_0$ rappresentino lo stesso numero. Senza perdita di generalità, possiamo assumere che³ $c_{k-1} \neq d_{k-1}$. L'ipotesi dice che

$$c_{k-1}b^{k-1} + c_{k-2}b^{k-2} + \dots c_1b^1 + c_0b^0 = d_{k-1}b^{k-1} + d_{k-2}b^{k-2} + \dots d_1b^1 + d_0b^0$$

cioè

$$(c_{k-1} - d_{k-1})b^{k-1} = (d_{k-2} - c_{k-2})b^{k-2} + \dots (d_1 - c_1)b^1 + (d_0 - c_0)b^0.$$

Sul lato destro dell'uguaglianza, i coefficienti $d_i - c_i$ sono tutti compresi fra $-(b - 1)$ e $b - 1$ (si tratta della differenza fra due cifre in base b); usando il Teorema 3, il valore della somma è compreso fra $-(b^{k-1} - 1)$ e $b^{k-1} - 1$. Sul lato sinistro dell'uguaglianza, viceversa, compare un multiplo di b^{k-1} . L'unico multiplo di b^{k-1} compreso fra $-(b^{k-1} - 1)$ e $b^{k-1} - 1$ è 0, e quindi $c_{k-1} = d_{k-1}$: contro l'ipotesi iniziale che $c_{k-1} \neq d_{k-1}$.

(c) Come abbiamo notato, usando k cifre si possono comporre b^k diversi numerali, e tutti (da (b)) rappresentano valori diversi; poiché (da (a)) i valori rappresentabili sono tutti compresi fra 0 e $b^k - 1$, e in questo intervallo ci sono in tutto b^k valori, ne segue che tutti e soli i valori dell'intervallo sono rappresentabili, e inoltre (necessariamente) lo sono in modo unico. \square

Detto in altri termini:

³Se così non fosse, basterebbe partire nella dimostrazione dalla prima cifra in cui i due numerali differiscono.

Corollario 5. *I numerali in base b con k cifre sono un sistema di rappresentazione completo e univoco per l'insieme $\{0, 1, \dots, b^k - 1\}$.*

Il precedente teorema ha come conseguenza che ogni numero naturale ha una rappresentazione in base b (usando un numero sufficientemente grande di cifre): cioè, il sistema di numerazione posizionale in base b è completo (per \mathbf{N}). Il sistema non è, strettamente parlando, univoco, poiché lo stesso numero può essere rappresentato da numerali che differiscono per il numero di zeri iniziali: ad esempio, i numerali 15, 015, 0000015, ecc. rappresentano tutti lo stesso numero. Nell'uso comune, però, si considerano come numerali validi solo quelli che non contengono zeri iniziali (ad eccezione, naturalmente, del numerale 0): tali numerali vengono chiamati *normalizzati*.

Teorema 6. *Il sistema di numerazione posizionale in base b è completo e, se si considerano solo i numerali normalizzati, è univoco.*

2.1.4 Conversioni da base decimale a base b

La definizione stessa di sistema di numerazione posizionale consente facilmente (come abbiamo visto nella sezione 2.1.2) di calcolare quale sia il numero rappresentato da un dato numerale. Il problema inverso, cioè determinare quale numerale rappresenta un certo numero (in una base fissata), si può risolvere con il seguente algoritmo:

1. Dividere x per b , scrivendo separatamente risultato (intero) e resto della divisione: il resto, ovviamente, sarà compreso fra 0 e $b - 1$;
2. Ripetere il procedimento usando ogni volta come x il risultato della divisione precedente;
3. Smettere non appena il risultato della divisione sia 0;
4. Il numerale è costituito dalla sequenza dei resti ottenuti, letta “al contrario”: cioè, la prima cifra (a sinistra) è l'ultimo resto, la seconda cifra è il penultimo e così via.

Illustriamo l'algoritmo attraverso alcuni esempi.

Alcuni esempi: da base 10 a base b .

1. quale numerale rappresenta il numero 167 in base 2?

167	83	resto 1
83	41	resto 1
41	20	resto 1
20	10	resto 0
10	5	resto 0
5	2	resto 1
2	1	resto 0
1	0	resto 1

Quindi $10100111_2 = 167$. Per verifica, convertiamo questo numerale:

1	0	1	0	0	1	1	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Quindi $10100111_2 = 2^7 + 2^5 + 2^2 + 2^1 + 2^0 = 128 + 32 + 4 + 2 + 1 = 167$.

2. quale numerale rappresenta il numero 167 in base 7?

167	23	resto 6
23	3	resto 2
3	0	resto 3

Quindi $326_7 = 167$.

3. quale numerale rappresenta il numero 1800 in base 13?

1800	138	resto 6
138	10	resto 8
10	0	resto 10(=A)

Quindi $A86_{13} = 1800$.

4. quale numerale rappresenta il numero 1800 in base 16?

1800	112	resto 8
112	7	resto 0
7	0	resto 7

Quindi $708_{16} = 1800$.

Alcuni esempi: da base b a base b' . I seguenti esempi mostrano come si converte da una base all'altra: il modo più semplice, in realtà, è passare attraverso la base 10.

1. scrivere 176_9 in base 7.

$$\begin{array}{|c|c|c|} \hline 1 & 7 & 6 \\ \hline 9^2 & 9^1 & 9^0 \\ \hline \end{array}$$

Quindi $176_9 = 1 \cdot 9^2 + 7 \cdot 9^1 + 6 \cdot 9^0 = 1 \cdot 81 + 7 \cdot 9 + 6 \cdot 1 = 81 + 63 + 6 = 150$.
Scrivendo 150 in base 7, abbiamo:

$$\begin{array}{r|l} 150 & 21 \quad \text{resto 3} \\ 21 & 3 \quad \text{resto 0} \\ 3 & 0 \quad \text{resto 3} \end{array}$$

Cioè, $303_7 = 150 = 176_9$.

2. scrivere $A01_{16}$ in base 11.

$$\begin{array}{|c|c|c|} \hline A & 0 & 1 \\ \hline 16^2 & 16^1 & 16^0 \\ \hline \end{array}$$

Quindi $A01_{16} = 10 \cdot 16^2 + 1 \cdot 16^0 = 2560 + 1 = 2561$. Scrivendo 2561 in base 11, abbiamo:

$$\begin{array}{r|l} 2561 & 232 \quad \text{resto 9} \\ 232 & 21 \quad \text{resto 1} \\ 21 & 1 \quad \text{resto 10(=A)} \\ 1 & 0 \quad \text{resto 1} \end{array}$$

Cioè, $1A19_{11} = 2561 = A01_{16}$.

Alcuni esempi: numerali in base incognita. Come ultimi esempi, consideriamo il problema di stabilire in che base è scritto un certo numerale.

1. trovare la base b per cui $103_b = 52$.

Deve essere $1 \cdot b^2 + 3 \cdot b^0 = 52$, cioè $b^2 + 3 = 52$, ovvero $b^2 = 49$, da cui $b = 7$.

2. trovare la base b per cui $342_b = 97$.

Deve essere $3b^2 + 4b + 2 = 97$, cioè $3b^2 + 4b - 95 = 0$. Il discriminante è $4^2 - 4 \cdot 3 \cdot (-95) = 16 + 1140 = 1156$; quindi le soluzioni dell'equazione sono $b_{1,2} = (-4 \pm \sqrt{1156})/6 = (-4 \pm 34)/6$. Scartando la soluzione negativa (che è inaccettabile) si ha $b = 30/6 = 5$.

3. trovare la base b per cui $534_b = 2343_5$.

Si ha $2343_5 = 2 \cdot 5^3 + 3 \cdot 5^2 + 4 \cdot 5^1 + 3 \cdot 5^0 = 2 \cdot 125 + 3 \cdot 25 + 4 \cdot 5 + 3 \cdot 1 = 250 + 75 + 20 + 3 = 348$. Cerchiamo quindi b per cui $534_b = 348$, cioè $5b^2 + 3b + 4 = 348$, ovvero $5b^2 + 3b - 344 = 0$. Le soluzioni sono $b_{1,2} = (-3 \pm \sqrt{3^2 + 4 \cdot 5 \cdot 344})/10 = (-3 \pm \sqrt{6889})/10 = (-3 \pm 83)/10$. L'unica soluzione positiva è $b = 80/10 = 8$.

2.1.5 Conversioni rapide per le basi 2, 8, 16

Per vari motivi storici e pratici, in informatica si usano spesso le basi 2, 8 e 16. Esistono regole rapide per la conversione fra queste basi, che riassumeremo in questa sezione.

Conversione da base 8 a base 2. Per convertire da base 8 a base 2, è sufficiente convertire separatamente ogni singola cifra, avendo cura di *aggiungere eventualmente degli zeri all'inizio in modo che ogni cifra ottale venga convertita in esattamente tre cifre binarie*. Ad esempio, per convertire 3721_8 basta notare che $3 = 011_2$, $7 = 111_2$, $2 = 010_2$ e $1 = 001_2$. Quindi $3721_8 = 11111010001_2$.

Conversione da base 16 a base 2. Per convertire da base 16 a base 2 si procede in modo analogo, ma *convertendo ogni cifra esadecimale in esattamente quattro cifre binarie*. Ad esempio, per convertire $D3A_{16}$ basta notare che $D_{16} = 13 = 1101_2$, $3 = 0011_2$, $A_{16} = 10 = 1010_2$, quindi $D3A_{16} = 110100111010_2$.

Conversione da base 2 a base 8. Per convertire da base 2 a base 8, si raggruppano le cifre binarie a gruppi di 3 *a partire da destra*, e si converte ogni gruppo in una singola cifra ottale. Ad esempio, per convertire 10011010011_2 si raggruppano le cifre come $10\ 011\ 010\ 011_2$ e quindi, essendo $011_2 = 3$ e $010_2 = 2$ si ha $10011010011_2 = 2323_8$.

Conversione da base 2 a base 16. Per convertire da base 2 a base 16 si procede analogamente, ma raggruppando le cifre binarie a gruppi di 4. Ad esempio, 10110101101_2 si scrive come $101\ 1010\ 1101_2 = 5AD_{16}$ (essendo $101_2 = 5$, $1010_2 = 10 = A_{16}$ e $1101_2 = 13 = D_{16}$).

2.1.6 Operazioni in base b

Uno dei principali motivi che hanno determinato il successo dei sistemi di numerazione posizionale sta nella facilità con cui si possono eseguire calcoli manipolando direttamente i numerali. Tutti abbiamo imparato, alle scuole elementari, a calcolare le quattro operazioni aritmetiche fondamentali (somma, sottrazione, moltiplicazione e divisione) manipolando i numeri in base 10. In effetti, queste tecniche (più precisamente: questi *algoritmi*) possono essere usati per fare calcoli in una base arbitraria. L'unica accortezza è che i riporti si fanno non più quando si eccede il 10 ma quando si eccede b .

Vediamo qualche esempio, limitandoci alle somme e alle sottrazioni.

1. calcolate $16_8 + 154_8$.

$$\begin{array}{r} 1\ 6\ + \\ 1\ 5\ 4 \\ \hline \end{array} \implies \begin{array}{r} 1^1\ 6\ + \\ 1\ 5\ 4 \\ \hline 2 \end{array} \implies \begin{array}{r} 1^1\ 6\ + \\ 1\ 5\ 4 \\ \hline 7\ 2 \end{array} \implies \begin{array}{r} 1^1\ 6\ + \\ 1\ 5\ 4 \\ \hline 1\ 7\ 2 \end{array}$$

Si parte dalla cifra più a destra. $6 + 4 = 10$, che sarebbe $8 + 2$: quindi scriviamo 2 con riporto di 1; $1 + 1 + 5 = 7$, senza riporto.

2. calcolate $37_{11} + 59_{11}$:

$$\begin{array}{r} 3\ 7\ + \\ 5\ 9 \\ \hline \end{array} \implies \begin{array}{r} 3^1\ 7\ + \\ 5\ 9 \\ \hline 5 \end{array} \implies \begin{array}{r} 3^1\ 7\ + \\ 5\ 9 \\ \hline 9\ 5 \end{array}$$

3. calcolate $103_8 - 57_8$:

$$\begin{array}{r} 1\ 0\ 3\ - \\ 5\ 7 \\ \hline \end{array} \implies \begin{array}{r} 1^{(-1)}\ 0^7\ 3\ - \\ 5\ 7 \\ \hline 4 \end{array} \implies \begin{array}{r} 1^{(-1)}\ 0^7\ 3\ - \\ 5\ 7 \\ \hline 2\ 4 \end{array}$$

Si parte dalla cifra più a destra. $3 - 7$ non si può fare: dobbiamo andare in prestito; siccome la cifra successiva è uno zero, andiamo di nuovo in prestito dalla cifra che segue: togliamo 1 dalla cifra da cui siamo andati in prestito, e diamo 7 unità alla seconda cifra (lo zero) e otto unità alla

cifra da cui eravamo partiti (in generale, daremo $b - 1$ unità a tutti gli zeri che abbiamo dovuto sorpassare, e b unità alla cifra da cui siamo partiti). Ora abbiamo $3(+8) - 7 = 11 - 7 = 4$. Quando passiamo alla seconda cifra, abbiamo $0(+7) - 5 = 2$, senza bisogno di ulteriori prestiti. Per l'ultima cifra, si ha $1(-1) - 0 = 0$.

2.2 Computer e rappresentazione digitale delle informazioni

Al di là dell'interesse intrinseco dei sistemi di rappresentazione dei numeri in base b , l'importanza della rappresentazione binaria in informatica risiede nel fatto che tutti i supporti di memoria sono costituiti da una sequenza di elementi atomici, detti *bit* (acronimo di “Binary digIT”, cioè “cifra binaria”) ciascuno dei quali può assumere solo due stati, chiamati astrattamente⁴ 0 e 1.

Qualunque informazione o dato debba essere immagazzinato in memoria va convertito in una opportuna sequenza binaria. Questo vale sia per i dati più semplici (numeri, caratteri ecc.) sia per i dati più sofisticati (immagini, suoni, filmati ecc.). Per ogni specifico tipo di dato, occorre stabilire come quel dato venga convertito e rappresentato sotto forma di una sequenza di bit. Questo processo viene chiamato *rappresentazione digitale dell'informazione*, e in genere si può realizzare in molti modi diversi; cioè, per lo stesso tipo di dati esisteranno molti diversi modi per rappresentarli digitalmente, o, se volete, molte diverse convenzioni che consentano di tradurre quel particolare tipo di dati in una sequenza di bit. A volte questi vengono chiamati *formati* di rappresentazione; ad esempio, per le immagini i formati più comuni sono JPG, GIF, PNG ecc.: ognuno di questi formati (o, più precisamente, famiglie di formati) stabilisce esattamente come una certa immagine debba essere rappresentata come sequenza di bit.

È importante notare che i formati di rappresentazione delle informazioni possono rappresentare il dato in modo fedele (in gergo, “lossless”), cioè, in modo tale che in seguito sia possibile risalire esattamente al dato rap-

⁴Come fisicamente sia realizzato ogni singolo bit dipende dalla tecnologia con cui è realizzato il supporto di memoria che stiamo considerando: ad esempio, nel caso di un supporto ottico (come, ad esempio, un CD-ROM o un DVD), si può trattare di una piccola zona di una superficie plastica riflettente che può riflettere un raggio luminoso secondo due diversi angoli; oppure, nel caso di un supporto magnetico (come un floppy disk o un disco fisso), può essere un frammento di un materiale magnetizzabile che può essere magnetizzato in due modi diversi. Questi aspetti tecnologici sono irrilevanti per la nostra discussione: ciò che conta è solo che la memoria è costituita da una sequenza di cellette, ciascuna delle quali può assumere solo due stati possibili.



Figura 1: Un'immagine a toni di grigio.

presentato, oppure in modo approssimato (in gergo, “lossy”), cioè, perdendo alcune informazioni e rendendo impossibile ricostruire esattamente il dato rappresentato.

Per maggiore concretezza, mostreremo ora un esempio.

2.2.1 Un esempio: rappresentazione di immagini a toni di grigio

Supponiamo di voler stabilire un nostro formato per la rappresentazione di immagini. Per semplicità, consideriamo immagini a toni di grigio; in Figura 1 vedete l'immagine originale che considereremo come esempio⁵.

Il primo passo per rappresentare digitalmente l'immagine consiste nel sovrapporre (idealmente) una griglia sull'immagine, come in Figura 2: la dimensione (numero di righe e di colonne) della griglia verrà chiamata *risoluzione*. Supponiamo, per fissare le idee, che si usi una griglia di 300x400 celle.

Ogni singola cella in cui risulta suddivisa l'immagine contiene un piccolo frammento dell'immagine stessa; se ci concentriamo su una cella specifica, essa conterrà naturalmente zone con toni di grigio diversi. A questo punto (ed è questa la prima approssimazione che introduciamo nella rappresentazione) calcoliamo il tono di grigio medio della cella e assumiamo che l'intera cella abbia esattamente quel colore. Il tono di grigio è scelto (ed è la seconda approssimazione) in una scala contenente un numero finito di toni di grigio: per fissare le idee, supponiamo di disporre di una scala con 256 toni di grigio,

⁵Notate che il processo descritto in questa sezione non ha alcuna pretesa di essere realistico o di corrispondere a qualche reale rappresentazione di immagini a toni di grigio, ma ha l'unico obiettivo di presentare concretamente il problema della rappresentazione digitale dei dati.



Figura 2: L'immagine di Figura 1 a cui è stata sovrapposta una griglia.

numerati da 0 a 255. Lo 0 corrisponde al bianco, il 255 al nero, e i livelli intermedi a toni di grigio via via più scuri.

Quindi, l'intera immagine può essere rappresentata (sebbene con le due approssimazioni introdotte sopra) come una sequenza di $300 \times 400 = 120000$ valori, ciascuno compreso fra 0 e 255 e rappresentante il tono di grigio di una cella. Poiché con 8 cifre binarie si possono rappresentare esattamente tutti e soli i numeri fra 0 e $2^8 - 1 = 256 - 1 = 255$ (Corollario 5), basteranno 8 bit per ogni cella per esprimere il tono di grigio di quella cella. In tutto, cioè, l'immagine verrà rappresentata da $8 \times 120000 = 960000$ bit.

2.2.2 Rappresentazione dei naturali; riporto e overflow

L'esempio descritto serve a illustrare come anche i tipi di dati più complessi possano essere convertiti in sequenze di bit, a discapito, in qualche caso, della fedeltà della rappresentazione. Tornando a tipi di dati più semplici, la discussione relativa alla rappresentazione dei numeri naturali in base b dovrebbe essere sufficiente a spiegare come si possa rappresentare un numero naturale in memoria. Ciò che accade, di solito, è che si fissa un certo numero k di bit, e si usano quei bit per rappresentare un numero naturale. Ovviamente, non potremo rappresentare *tutti* i numeri naturali ma solo quelli compresi fra 0 e $2^k - 1$. La scelta di k è legata quindi al massimo numero naturale che vogliamo rappresentare; ad esempio, se sappiamo di voler rappresentare valori non superiori a 1 000 000, dovremo usare $k = 20$ bit ($2^{20} = 1\,048\,576$; invece $2^{19} = 524\,288$, quindi 19 bit non sarebbero sufficienti).

Le operazioni aritmetiche verranno effettuate dall'unità aritmetico-logica (detta anche ALU, la parte della CPU che si occupa di svolgere le operazioni aritmetiche), usando essenzialmente gli stessi algoritmi che abbiamo visto

nella Sezione 2.1.6; un'osservazione importante, a questo proposito, riguarda il problema del riporto. Supponete di avere due numeri naturali contenuti in memoria, ciascuno in una sequenza di k bit, e supponete di voler calcolare la somma dei due naturali, mettendo il risultato in una terza sequenza di k bit.

Istruite a questo scopo l'ALU affinché effettui la somma: i due valori saranno necessariamente compresi fra 0 e $2^k - 1$, e quindi la loro somma sarà compresa fra 0 e $2^{k+1} - 2$; ora, se la somma è minore di 2^k , il risultato è rappresentabile in k bit, e quindi non ci sono problemi, ma se la somma è maggiore o uguale di 2^k il risultato non è più rappresentabile in k bit.

Vediamo un esempio semplice, supponendo che $k = 8$, cioè che si usino 8 bit per rappresentare i dati. Supponiamo che il primo gruppo di 8 bit contenga il valore 216, mentre il secondo gruppo di 8 bit contenga 98: indichiamo con ? il contenuto degli 8 bit in cui scriveremo il risultato (in quanto non siamo interessati al loro contenuto iniziale).

$$\begin{array}{r}
 216 \quad + \\
 98 \\
 \hline
 314
 \end{array}
 \quad
 \begin{array}{|c|c|c|c|c|c|c|c|}
 \hline
 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
 \hline
 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
 \hline
 ? & ? & ? & ? & ? & ? & ? & ? \\
 \hline
 \end{array}$$

Notate che il risultato della somma, 314, non è rappresentabile in 8 bit (il massimo naturale rappresentabile con 8 bit è $2^8 - 1 = 255$). In effetti, svolgendo la somma rimane un riporto:

$$\text{Riporto: } \boxed{1}
 \quad
 \begin{array}{|c|c|c|c|c|c|c|c|}
 \hline
 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
 \hline
 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
 \hline
 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
 \hline
 \end{array}$$

Il risultato *sarebbe* 100111010, ma il bit di riporto non è memorizzabile nei k bit. Se guardiamo gli 8 bit del risultato, essi non contengono il valore corretto, 314 (né potrebbero contenere questo valore, visto che è troppo grande), ma contengono invece 58. Tale situazione viene chiamata *overflow*, e si verifica tutte le volte che il risultato di un'operazione aritmetica non è rappresentabile nel numero di bit a nostra disposizione.

La CPU contiene al suo interno un bit, detto *bit di riporto* (o, in inglese, *carry*) che indica il riporto causato dall'ultima operazione aritmetica effettuata: se il bit di riporto è 1, sappiamo che l'ultima operazione ha dato luogo a un overflow e quindi che il risultato immagazzinato non è corretto (nel senso che manca di un bit).

Dopo ogni operazione aritmetica, dovremmo verificare il contenuto del bit di riporto per evitare di interpretare erroneamente il risultato oppure, in alternativa, dovremmo controllare i valori degli operandi *prima* di effettuare

l'operazione e prendere delle decisioni (per esempio, segnalare un errore) se scopriamo che l'effettuazione dell'operazione darà luogo a un overflow. Per evitare questo tipo di controlli, ove possibile sarà sempre meglio scegliere k molto più grande di quanto ci servirebbe in effetti, in modo che non si producano overflow nel manipolare i dati. È comunque importante sapere che il problema dell'overflow esiste, e che quando un programma produce risultati diversi da quelli attesi una delle cause possibili è la presenza di overflow nei calcoli.

2.3 Rappresentazione degli interi

In questa sezione ci occuperemo del problema di rappresentare interi *con segno*; sebbene sia possibile, in linea di principio, illustrare questo concetto per una base generica, ci limiteremo d'ora in avanti a considerare la base binaria, che è quella con maggiori applicazioni pratiche.

2.3.1 Rappresentazione mediante modulo e segno

La prima tecnica di rappresentazione per gli interi consiste nell'utilizzare un bit (detto *bit di segno*) per rappresentare il segno dell'intero (ad esempio, 0 per il segno positivo e 1 per il segno negativo) e i bit restanti per rappresentare il *modulo* (cioè, il valore assoluto) del numero. Quindi, se si hanno a disposizione solo k cifre binarie, $k - 1$ verranno utilizzate per rappresentare il modulo, consentendo di rappresentare i moduli da 0 a $2^{k-1} - 1$ (Corollario 5).

Teorema 7. *Nella rappresentazione mediante modulo e segno, avendo a disposizione $k > 1$ bit, si possono rappresentare tutti i valori da $-(2^{k-1} - 1)$ a $2^{k-1} - 1$. Tutti i valori hanno una rappresentazione unica, tranne lo 0 che ha due rappresentazioni.*

Il fatto che lo 0 abbia due rappresentazioni segue ovviamente dal fatto che, se il modulo è 0, il valore è 0 indipendentemente dal bit di segno.

Alcuni esempi.

1. Qual è la rappresentazione del numero 15 su 6 bit in modulo e segno?

Convertiamo 15 in binario:

15	7	resto 1
	7	resto 1
	3	resto 1
	1	resto 1
	1	resto 0

Ora, ricordiamo che dei 6 bit a nostra disposizione, il primo va usato per il segno (0, essendo il segno positivo) e i restanti 5 per il modulo:

0	0	1	1	1	1
---	---	---	---	---	---

2. Qual è la rappresentazione di -123 su 8 bit in modulo e segno?

Convertiamo 123 in binario:

123	61	resto 1
	61	resto 1
	30	resto 0
	15	resto 1
	7	resto 1
	3	resto 1
	1	resto 0

Quindi:

1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---

2.3.2 Rappresentazione in complemento a due

Sebbene in linea di principio la rappresentazione mediante modulo e segno appaia essere molto naturale, presenta difetti che la rendono di fatto molto poco utilizzata in pratica. In particolare, realizzare circuiti per la manipolazione aritmetica di numeri interi rappresentati mediante modulo e segno risulta estremamente complesso e dispendioso. È questo problema a rendere molto più comune la rappresentazione cosiddetta *in complemento a due*, che presenteremo ora.

La rappresentazione in complemento a due con $k > 1$ bit è una semplice rappresentazione posizionale, con la sola differenza che il bit di sinistra, che nella normale rappresentazione posizionale corrisponderebbe alla potenza 2^{k-1} , viene invece interpretato come -2^{k-1} ; quindi, ad esempio, il numerale $c_{k-1} \dots c_3 c_2 c_1 c_0$ viene interpretato così:

c_{k-1}	c_{k-2}	\dots	c_3	c_2	c_1	c_0
-2^{k-1}	2^{k-2}		2^3	2^2	2^1	2^0

Cioè, il suo valore è $-c_{k-1}2^{k-1} + c_{k-2}2^{k-2} + \dots + c_32^3 + c_22^2 + c_12^1 + c_0$. Vediamo quali caratteristiche ha questo sistema di rappresentazione:

Teorema 8. *Consideriamo la rappresentazione in complemento a due con $k > 1$ bit. Allora:*

1. *il valore rappresentato è negativo se e solo se il primo bit (di sinistra) è 1;*
2. *i valori rappresentabili sono tutti e soli i valori fra -2^{k-1} e $2^{k-1} - 1$, e ciascuno è rappresentabile in un solo modo.*

Dimostrazione. (1) Se il primo bit è 0, il valore rappresentato è ovviamente maggiore o uguale a zero. Supponiamo che il primo bit sia 1; il massimo valore possibile in questo caso si ottiene facendo in modo che anche tutti gli altri $k-1$ bit siano a 1, ma anche così otterremmo come valore $-2^{k-1} + 2^{k-2} + \dots + 2^2 + 2^1 + 2^0$. Applicando il Teorema 3 (con $b = 2$) la somma indicata vale $-2^{k-1} + 2^{k-1} - 1 = -1$: quindi, se il primo bit è 1 il valore rappresentato è per forza negativo.

(2) Il più piccolo valore rappresentabile è -2^{k-1} (si ottiene mettendo a 1 il primo bit e a 0 tutti gli altri); il più grande valore rappresentabile è $2^{k-1} - 1$ (il primo bit a 0 e tutti gli altri a 1). Per mostrare che non esistono valori con due rappresentazioni distinguiamo due casi: per quanto riguarda i valori maggiori o uguali a zero, ciò è ovvio (la rappresentazione è, a parte il primo bit, identica alla rappresentazione dei numeri naturali su $k-1$ bit); per i valori negativi, supponete che i numerali $c_{k-1}c_{k-2}\dots c_1c_0$ e $c'_{k-1}c'_{k-2}\dots c'_1c'_0$ rappresentino lo stesso numero negativo (cioè, $c_{k-1} = c'_{k-1} = 1$). Vorrebbe dire che

$$-c_{k-1}2^{k-1} + c_{k-2}2^{k-2} + \dots + c_12^1 + c_0 = -c'_{k-1}2^{k-1} + c'_{k-2}2^{k-2} + \dots + c'_12^1 + c'_0.$$

Essendo $c_{k-1} = c'_{k-1}$ si ha

$$c_{k-2}2^{k-2} + \dots + c_32^3 + c_22^2 + c_12^1 + c_0 = c'_{k-2}2^{k-2} + \dots + c'_32^3 + c'_22^2 + c'_12^1 + c'_0$$

che vorrebbe dire che $c_{k-2}\dots c_1c_0$ e $c'_{k-2}\dots c'_1c'_0$ sono due numerali che rappresentano lo stesso numero naturale su $k-1$ bit, il che contraddirebbe il Corollario 5. \square

Alcuni esempi.

1. Che numero è rappresentato in complemento a due in questa sequenza di 8 bit?

1	1	1	0	1	1	1	1
---	---	---	---	---	---	---	---

Si tratta di $-2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0 = -128 + 64 + 32 + 8 + 4 + 2 + 1 = -17$.

2. Che numero è rappresentato in complemento a due in questa sequenza di 5 bit?

1	0	0	0	1
---	---	---	---	---

Si tratta di $-2^4 + 2^0 = -16 + 1 = -15$.

3. Che numero è rappresentato in complemento a due in questa sequenza di 5 bit?

0	1	0	0	1
---	---	---	---	---

Si tratta di $2^3 + 2^0 = 8 + 1 = 9$. In effetti, per i numeri in complemento a due in cui il primo bit sia 0, la rappresentazione non è in nulla diversa dall'usuale rappresentazione del numero come numero naturale (eccettuato per il bit iniziale a 0).

Regola pratica di rappresentazione. Gli esempi indicati mostrano come sia semplice comprendere quale numero sia rappresentato in complemento a due in una certa sequenza di bit. Un po' meno ovvio è comprendere, dato un numero x e un intero $k > 1$, come rappresentare x in complemento a due su k bit. Se $x \geq 0$, la cosa è, come dicevamo, ovvia: si tratta semplicemente di rappresentare x (come numero naturale) su $k - 1$ bit, facendo precedere la rappresentazione da uno 0. Se $x < 0$, invece, conviene rappresentare $x + 2^{k-1}$ su $k - 1$ bit come numero naturale facendo precedere tale rappresentazione da un 1.

Alcuni esempi.

1. Qual è la rappresentazione di -123 su 8 bit in complemento a due?

Convertiamo $-123 + 2^{8-1} = -123 + 2^7 = -123 + 128 = 5$ in binario:

$$\begin{array}{r|l} 5 & 2 \quad \text{resto 1} \\ 2 & 1 \quad \text{resto 0} \\ 1 & 0 \quad \text{resto 1} \end{array}$$

Ovvero (su 7 bit) 0000101. Quindi la rappresentazione è:

1	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

2. Qual è la rappresentazione di 123 su 8 bit in complemento a due?

Convertiamo 123 in binario:

123	61	resto 1
61	30	resto 1
30	15	resto 0
15	7	resto 1
7	3	resto 1
3	1	resto 1
1	0	resto 1

Quindi:

0	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---

Una regola alternativa. Esiste un modo alternativo ed equivalente per ottenere lo stesso risultato. Supponete di voler rappresentare il valore *negativo* $-x$ su $k > 1$ bit in complemento a 2 (consideriamo solo il caso di numeri negativi, perché il caso dei numeri maggiori o uguali a zero è, come dicevamo, banale). Calcolate la rappresentazione del numero naturale $x - 1$ su $k - 1$ bit; invertite tutti i bit del risultato (cioè, trasformate gli 0 in 1 e viceversa) e premettete un 1 al risultato.

Il motivo che giustifica questo metodo è il seguente: supponete che $c_{k-2}c_{k-3}\dots c_1c_0$ sia la rappresentazione di $x - 1$ su $k - 1$ bit; ciò significa che $c_{k-2}2^{k-2} + \dots + c_02^0 = x - 1$. Seguendo la regola descritta, $1\bar{c}_{k-2}\dots\bar{c}_1\bar{c}_0$ è la rappresentazione in complemento a 2 di $-x$ (qui, abbiamo indicato con \bar{c}_i l'inverso di c_i). Per verificare che ciò è vero, osserviamo che sommando $\bar{c}_{k-2}\dots\bar{c}_1\bar{c}_0$ con $c_{k-2}c_{k-3}\dots c_1c_0$ otteniamo $11\dots 11$, che corrisponde alla rappresentazione di $2^{k-1} - 1$. Quindi $\bar{c}_{k-2}\dots\bar{c}_1\bar{c}_0$ rappresenta $2^{k-1} - 1 - (c_{k-2}2^{k-2} + \dots + c_02^0) = 2^{k-1} - 1 - (x - 1) = 2^{k-1} - x$. Ma allora la sequenza $1\bar{c}_{k-2}\dots\bar{c}_1\bar{c}_0$ rappresenta il valore $-2^{k-1} + 2^{k-1} - x = -x$, come volevasi dimostrare.

Alcuni esempi.

1. Qual è la rappresentazione di -123 su 8 bit in complemento a due?

Convertiamo $123 - 1 = 122$ in binario:

Sul lato destro abbiamo indicato i valori contenuti nei tre gruppi di bit, interpretando ciascuno come se contenesse un numero naturale. La somma, in questo esempio, dà luogo a un riporto (e in effetti il risultato *non* è corretto, cioè non è $216 + 98 = 314$, poiché quest'ultimo numero non è rappresentabile su 8 bit).

Ora cambiamo l'interpretazione dei dati, cioè *interpretiamo i dati come se fossero rappresentati in complemento a due*:

Riporto:	1		1	1	0	1	1	0	0	0	-40
			0	1	1	0	0	0	1	0	98
			0	0	1	1	1	0	1	0	58

Con questa nuova interpretazione, la somma (algebrica) risulta corretta ($-40 + 98 = 58$). È un caso? E che significato ha il bit di riporto con questa nuova interpretazione?

Il seguente teorema, di cui omettiamo la dimostrazione, risponde a queste domande:

Teorema 9. *Siano $C = c_{k-1} \dots c_0$ e $D = d_{k-1} \dots d_0$ due sequenze di k bit che, interpretate in complemento a due, rappresentano i numeri interi x e y , rispettivamente. Sia $E = e_{k-1} \dots e_0$ la sequenza di k bit ottenuta sommando C e D come se rappresentassero due numeri naturali, siano r il bit di riporto finale e s il bit di riporto ottenuto prima di aver sommato l'ultima cifra (cioè dopo aver sommando $c_{k-2} \dots c_0$ con $d_{k-2} \dots d_0$). Allora:*

- *se $r = s$, allora $x + y$ è compreso fra -2^{k-1} e $2^{k-1} - 1$ (cioè, è rappresentabile in complemento a due su k bit), e $e_{k-1} \dots e_0$ contiene la rappresentazione in complemento a due di $x + y$;*
- *se $r \neq s$, allora $x + y$ non è compreso fra -2^{k-1} e $2^{k-1} - 1$, cioè si è verificato un overflow.*

In sintesi, ciò che il teorema dice è che date due generiche sequenze di k bit, l'algoritmo della somma (e ciò vale anche per gli algoritmi delle altre operazioni aritmetiche) funziona, in assenza di overflow, sia che si interpretino le sequenze come rappresentanti numeri naturali, sia che si interpretino come rappresentazioni in complemento a due di numeri interi. L'unica differenza si ha nel verificare l'overflow: nel caso dei numeri naturali, si ha overflow se e solo se alla fine il bit di riporto è a 1; nel caso dei numeri in complemento a due, si ha overflow se e solo se il bit di riporto finale ha un valore diverso dal bit di riporto prima di sommare l'ultima cifra.

2.4 Rappresentazione di numeri razionali

Fin qui ci siamo occupati della rappresentazione di numeri interi, con e senza segno, e abbiamo discusso vantaggi e svantaggi delle varie possibili soluzioni. Ci occuperemo ora della rappresentazione di numeri non interi: per farlo, converrà estendere la nozione di sistema di numerazione posizionale.

2.4.1 Sistemi di numerazione posizionale generalizzati

Come abbiamo visto, in un sistema di numerazione posizionale il valore di ogni cifra viene moltiplicato per un fattore che dipende dalla posizione che la cifra occupa; tale fattore è una potenza della base b scelta, e l'esponente è 0 per la cifra che si trova più a destra, 1 per quella immediatamente alla sua sinistra e così via. In effetti, la prima cifra a sinistra viene chiamata "cifra più significativa" poiché è quella che viene moltiplicata per il fattore più grande.

Questo sistema di numerazione si può estendere in modo da permetterci anche di rappresentare numeri razionali; per farlo, occorre considerare non solo le potenze *positive* della base, ma anche le potenze *negative*⁶, ottenendo quello che chiameremo *sistema di numerazione posizionale generalizzato in base b*:

...	Pos. k	...	Pos. 1	Pos. 0	,	Pos. -1	Pos. -2	...
	b^k		b^1	b^0		b^{-1}	b^{-2}	

Quindi, un numerale sarà rappresentato da due sequenze di cifre separate da una *virgola*: la sequenza di sinistra viene interpretata come un normale numero naturale in base b , la sequenza di destra viene interpretata analogamente, ma moltiplicando la prima cifra per b^{-1} , la seconda per b^{-2} e così via.

Alcuni esempi.

1. Che numero rappresenta il numerale $1011,11_2$?

1	0	1	1	,	1	1
2^3	2^2	2^1	2^0		2^{-1}	2^{-2}

Quindi $1011,11_2 = 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-2} = 8 + 2 + 1 + 1/2 + 1/4 = 11,75$.

2. Che numero rappresenta il numerale $730,0301_8$?

⁶Ricordiamo che b^{-k} significa $1/b^k$; ad esempio, $2^{-5} = 1/2^5 = 1/32$.

7	3	0	,	0	3	0	1
8^2	8^1	8^0		8^{-1}	8^{-2}	8^{-3}	8^{-4}

Quindi $730,0301_8 = 7 \cdot 8^2 + 3 \cdot 8^1 + 0 \cdot 8^0 + 0 \cdot 8^{-1} + 3 \cdot 8^{-2} + 0 \cdot 8^{-3} + 1 \cdot 8^{-4} = 472,047119140625$.

3. Che numero rappresenta il numerale A,AA_{16} ?

A	,	A	A
16^0		16^{-1}	16^{-2}

Quindi $A,AA_{16} = 10 \cdot 16^0 + 10 \cdot 16^{-1} + 10 \cdot 16^{-2} = 10,6640625$.

Osservazioni. Concludiamo con alcune osservazioni importanti che riguardano il sistema di numerazione generalizzato in base b .

- Per calcolare quale valore rappresenta una certa sequenza di cifre *con la virgola* si può anche procedere come segue: si sposta la sequenza a sinistra di un numero m sufficiente di cifre in modo da trovarsi la virgola all'estremo destro; si calcola il valore intero rappresentato dalla sequenza così ottenuta; si moltiplica il risultato per b^{-m} . Ad esempio, per calcolare il valore rappresentato da $730,0301_8$ si può considerare l'intero 7300301_8 (ottenuto spostando le cifre a sinistra di quattro posizioni) che rappresenta il valore 1933505 ; dunque $730,0301_8 = 1933505 \cdot 8^{-4} = 472,047119140625$.
- L'osservazione precedente ci permette di dedurre che i valori rappresentabili con questo sistema sono tutti e soli i numeri razionali che possono essere scritti nella forma x/b^k , i cosiddetti *razionali b-adici*. Essi hanno in effetti una rappresentazione finita unica. Viceversa, si può dimostrare che gli altri razionali sono rappresentabili solo usando *infinite cifre periodiche* dopo la virgola; è importante notare, a questo proposito, che i numeri che hanno una rappresentazione periodica dipendono dalla base. Ad esempio, il numero $1/5$ non è periodico in base 10, essendo $1/5 = 2/10$; infatti $0,2$ è la rappresentazione di $1/5$ in base 10 (e non è periodica). Se invece consideriamo la base 2, risulta $0,001100110011\dots_2 = 1/5$: cioè, in base 2 il numero $1/5$ è periodico.
- L'osservazione precedente ha delle importanti conseguenze pratiche. Se, come succede in pratica, si ha a disposizione solo un numero finito di cifre e il numero da rappresentare è periodico, ci si troverà a manipolare una sua approssimazione. Ad esempio, se si vuole rappresentare il numero $1/5 = 0,2$ in base 2 e si dispone solo di 7 cifre dopo la virgola, si rappresenterà il valore come $0,0011001_2$, il cui vero valore è $0,1953125$.

- Per quanto riguarda i numeri irrazionali, cioè i numeri che stanno nell'insieme $\mathbf{R} \setminus \mathbf{Q}$ (come, ad esempio, $\sqrt{2}$ o anche π), si può dimostrare che essi sono esattamente quelli che hanno (in qualunque base li si consideri) una rappresentazione infinita e non periodica. Così, ad esempio, $\sqrt{2} = 1,414213562373095048801688724209\dots_{10}$ oppure $\sqrt{2} = 1,011010100000100111100110011001111\dots_2$.

2.4.2 Rappresentazione in virgola fissa (fixed-point)

Un possibile sistema di rappresentazione dei numeri razionali ormai in disuso (se non per talune applicazioni di carattere finanziario) è la cosiddetta *rappresentazione in virgola fissa* (fixed-point). In pratica, per rappresentare un numero in virgola fissa si usano $k + h + 1$ bit: il primo bit si usa per il segno (0=segno positivo, 1=segno negativo), i seguenti k bit si usano per rappresentare la parte intera, e i restanti h bit per la parte decimale:

Segno	c_{k-1}	\dots	c_1	c_0	,	c_{-1}	c_{-2}	\dots	c_{-h}
	2^{k-1}		2^1	2^0		2^{-1}	2^{-2}		2^{-h}

È facile vedere che i valori rappresentabili in questo modo sono

$$\left\{ 0, \pm \frac{1}{2^h}, \pm \frac{2}{2^h}, \dots, \pm \left(2^k - \frac{1}{2^h} \right) \right\}$$

cioè, sono tutti i multipli positivi e negativi di 2^{-h} minori in valore assoluto di 2^k .

L'unico vantaggio di questa rappresentazione sta nel fatto che i valori rappresentabili sono equispaziati, cioè la distanza fra ciascuno e il successivo è costante; come nel caso della rappresentazione degli interi mediante modulo e segno, però, i vantaggi risultano molto meno rilevanti rispetto agli svantaggi, che hanno fatto preferire nella pratica l'applicazione della rappresentazione in virgola mobile.

2.4.3 Rappresentazione in virgola mobile (floating-point)

Nella *rappresentazione in virgola mobile* (floating-point), si usano⁷ $k + h + 1$ bit per rappresentare un numero; il primo bit, come nel caso della virgola fissa, indica il segno S (0 vuol dire segno positivo, cioè $S = 1$; 1 vuol dire segno negativo, cioè $S = -1$). I successivi k bit vengono interpretati come

⁷In effetti, gli standard per la rappresentazione in virgola mobile (ad esempio, l'IEEE 754 adottato, fra l'altro, da Java) sono assai più complessi di quanto indicato qui. Una descrizione completa, però, va al di là degli scopi di queste dispense.

la rappresentazione di un numero naturale E , chiamato *esponente*. Infine, i rimanenti h bit vengono interpretati anch'essi come la rappresentazione di un numero naturale M , chiamato *mantissa*. Il valore rappresentato è

$$S \cdot M \cdot 2^{E-w}$$

dove w è un valore fisso, chiamato *offset dell'esponente*.

Il più piccolo valore positivo rappresentabile (talvolta chiamato *epsilon*) è 2^{-w} , mentre il più grande valore positivo rappresentabile è $(2^h - 1) \cdot 2^{2^k - 1 - w}$. Fra questi due estremi, però, i valori rappresentabili non sono equispaziati ma sono, viceversa, più concentrati intorno allo zero, ed esponenzialmente meno concentrati man mano si procede verso i valori assoluti più grandi.

Anche nel caso dei valori floating-point si possono presentare situazioni di *overflow* (quando il risultato di un'operazione aritmetica è più grande, in valore assoluto, del massimo valore rappresentabile); ma si possono anche verificare situazioni di *underflow* (quando il risultato di un'operazione aritmetica è più piccolo dell'epsilon). Bisogna poi tener conto del fatto che la manipolazione di numeri floating-point è sempre soggetta ad approssimazioni, sia perché i soli numeri rappresentabili esattamente sono (alcun)i razionali diadici, cioè quelli esprimibili come $x/2^k$, sia perché l'effettuazione dei calcoli può comportare la perdita di cifre significative.

A questo proposito, conviene notare che l'ordine in cui vengono eseguiti i calcoli con numeri floating-point può influire grandemente sul risultato: ad esempio, se si dovesse dividere un numero floating-point x per un valore D e poi lo si dovesse moltiplicare per M , il risultato ottenuto potrebbe essere soggetto a gravi errori di approssimazione nel caso in cui D fosse molto grande rispetto a x (il che potrebbe addirittura causare un underflow); in tali casi moltiplicare prima per M e poi dividere per D potrebbe talvolta alleviare il problema.

Un esempio. Consideriamo il seguente esempio, con 4 bit per l'esponente, 6 per la mantissa e assumendo che $w = 6$.

±	Esponente				Mantissa					
1	0	0	1	1	0	1	0	1	1	1

Il segno è negativo ($S = -1$) essendo il bit di segno 1. L'esponente è 0011, cioè 3. La mantissa è 010111, cioè 23. Quindi il valore rappresentato è $-1 \cdot 23 \cdot 2^{3-6} = -0,375$.

2.5 Rappresentazione dei caratteri

L'ultimo tipo di dati elementare che prenderemo in considerazione è costituito dai caratteri. Il *carattere* è l'unità atomica in cui si decompone un testo; in effetti, un testo è semplicemente una sequenza di caratteri. La memorizzazione di documenti testuali è così frequente che il problema di come codificare i caratteri si è presentato molto presto in informatica.

Una delle prime soluzioni standardizzate è costituita dal cosiddetto *codice US-ASCII*, un sistema di codifica dei caratteri sviluppato dall'American Standard Code for Information Interchange. In questo sistema di codifica, ogni carattere è rappresentato da 7 bit. Poiché con 7 bit si possono ottenere $2^7 = 128$ combinazioni diverse (che, interpretate come numeri naturali, sono i numeri da 0 a 127), il codice US-ASCII comprende un repertorio di 128 caratteri. Il codice US-ASCII è mostrato in Tabella 1: per ognuno dei 128 codici (cioè, implicitamente, per ciascuna delle 128 combinazioni di 7 bit) è indicato il carattere corrispondente.

Notate che per “carattere” qui intendiamo il concetto astratto di carattere, da non confondere con l'aspetto grafico che il carattere assume (in gergo, *glifo*): lo stesso carattere può essere rappresentato da glifi molto diversi (per esempio, a seconda del font scelto), ed esistono caratteri che non hanno una rappresentazione grafica. Nel codice ASCII, in particolare, i primi 32 caratteri sono detti *caratteri di controllo* e non hanno una rappresentazione grafica, ma provocano degli effetti quando vengono visualizzati o stampati. Per esempio, il carattere 13 rappresenta il CR (Carriage Return, cioè ritorno del carrello) e ha l'effetto di riportare il carrello di stampa all'inizio della riga corrente, mentre il carattere 10 rappresenta il LF (Line Feed) e ha l'effetto di far avanzare di una riga il carrello. Naturalmente, il vero effetto prodotto dai codici di controllo dipende dalla periferica a cui essi vengono inviati (ad esempio, alcune stampanti potrebbero ignorare i CR/LF, oppure potrebbero dare luogo a comportamenti diversi quando ricevono un carattere di controllo).

ISO-8859. Con il diffondersi dell'uso degli elaboratori si è presentata l'esigenza di rappresentare nuovi caratteri che non appartengono al repertorio dell'US-ASCII. In particolare, ad esempio, l'assenza delle lettere accentate rende problematico l'utilizzo dell'US-ASCII per l'italiano, mentre l'assenza di opportuni segni diacritici (ad esempio L o å) rendono impossibile il suo uso per altre lingue europee.

Questo ha spinto l'ISO (International Organization for Standardization) a sviluppare delle estensioni dell'US-ASCII, note come ISO-8859. Si tratta di una serie di codici a 8 bit, in cui tutti i codici della forma $0xxxxxxx$ vengono interpretati come nell'US-ASCII, mentre i restanti codici della for-

Cod.	Car.	Cod.	Car.	Cod.	Car.	Cod.	Car.
0	NUL '\0'	32	SPACE	64	@	96	'
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL '\a'	39	'	71	G	103	g
8	BS '\b'	40	(72	H	104	h
9	HT '\t'	41)	73	I	105	i
10	LF '\n'	42	*	74	J	106	j
11	VT '\v'	43	+	75	K	107	k
12	FF '\f'	44	,	76	L	108	l
13	CR '\r'	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	—
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Tabella 1: Codice US-ASCII.

ma *1xxxxxx* (cioè, quelli corrispondenti ai numeri da 128 a 255) vengono interpretati come 128 altri caratteri.

Per i paesi dell'Europa occidentale, si è sviluppato l'ISO-8859-1 (anche chiamato Latin-1) che comprende, fra le altre, le lettere accentate, il simbolo della cediglia ecc. Ad esempio, il carattere à (a minuscola con accento grave) ha codice 224, il carattere á (a minuscola con accento acuto) ha codice 225 ecc.⁸

Contestualmente si sono sviluppate analoghe estensioni per altre zone d'Europa e del mondo, incompatibili con l'ISO-8859-1 se non per la parte comune a tutte (e rappresentata dall'US-ASCII, cioè dai primi 128 caratteri), per esempio l'ISO-8859-2 (per i paesi dell'Europa centrale e orientale), l'ISO-8859-5 (per il cirillico), l'ISO-8859-6 (per l'arabo), l'ISO-8859-8 (per l'ebraico) ecc.

ISO-10646 (o Unicode). Le ulteriori esigenze di internazionalizzazione che hanno caratterizzato gli ultimi anni (in particolare, e in modo esplosivo, con l'affermarsi di Internet) hanno spinto l'ISO a promuovere l'uso di nuovi standard più ampi, che fossero in grado di rappresentare caratteri in ogni lingua possibile e per ogni possibile esigenza. Questo sforzo ha dato origine a un nuovo standard, chiamato ISO-10646 o Unicode. Al momento, questo standard ha prodotto due diversi insiemi di caratteri: l'ISO-10646-1 (chiamato anche Basic Multilingual Plane) che è un'estensione dell'ISO-8859-1 e che comprende $2^{16} = 65534$ caratteri (di cui i primi 256 sono esattamente quelli dell'ISO-8859-1), e l'ISO-10646-2 che è un'estensione dell'ISO-10646-1 che ha spazio per circa un milione di altri caratteri⁹.

Lo standard ISO-10646 fissa il repertorio di caratteri e, per ciascun carattere, il codice (se ci limitiamo all'ISO-10646-1, un valore fra 0 e 65533) corrispondente: le tavole dei caratteri Unicode (contenenti tutti i caratteri disponibili in Unicode, suddivisi per lingua, ciascuno con il proprio codice, che è un numero fra 0 e 65533) possono essere trovate all'indirizzo <http://www.unicode.org/charts>.

Viceversa, lo standard ISO-10646 non è prescrittivo sul modo con cui ciascun codice vada rappresentato sottoforma di sequenza di bit. Se ci limitiamo all'ISO-10646-1, ogni carattere è rappresentabile usando 16 bit, e in particolare i caratteri della forma *00000000xxxxxxx* (cioè, quelli che hanno i primi 8 bit a zero) corrispondono al carattere *xxxxxxx* dell'ISO-8859-1. Questo

⁸Recentemente, l'ISO-8859-1 è stato lievemente modificato mediante la sostituzione di alcuni dei caratteri pre-esistenti con nuovi caratteri, in particolare il simbolo dell'Euro, prima non previsti; questa versione modificata prende il nome di ISO-8859-15.

⁹Al momento, l'ISO-10646-2 comprende pochi caratteri e ad uso estremamente specialistico, per cui spesso quando si parla di Unicode ci si riferisce solo all'ISO-10646-1.

modo di rappresentare i caratteri Unicode viene anche chiamato UCS-2 (Universal Character Set a 2 byte), ed è il modo che utilizza ad esempio Java per rappresentare internamente i caratteri. Ci sono però anche altri modi per rappresentare i caratteri dello standard ISO-10646, in particolare esiste una rappresentazione nota come UTF-8 che utilizza un numero *variabile* di bit per rappresentare ogni carattere; la spiegazione dettagliata di cosa significhi e come sia realizzata questa rappresentazione va al di là degli scopi di queste dispense.

3 Elementi di algebra di Boole e circuiti

In questa sezione intendiamo occuparci dei fondamenti di algebra di Boole, cioè della manipolazione di valori booleani. Il protagonista fondamentale di questa sezione sarà l'insieme $B = \{0, 1\}$, i cui due elementi 0 e 1 possono essere interpretati in molti modi diversi; per esempio:

- dato un qualunque sistema fisico che possa assumere solo due stati, possiamo identificare i due stati con 0 e 1: esempi di sistemi fisici di questo tipo sono una lampadina (che può essere solo “accesa” o “spenta”), un interruttore (che può essere solo “chiuso” o “aperto”), un filo su cui possa scorrere la corrente solo a due diversi livelli di tensione, un frammento di materiale che possa essere magnetizzato in due soli modi, una zona di superficie riflettente che possa riflettere un raggio luminoso secondo due soli angoli ecc.; tutti questi sistemi fisici sono, in linea di principio, utilizzabili per realizzare un dispositivo di memoria digitale, poiché ogni dispositivo di memoria digitale è semplicemente una sequenza di oggetti ciascuno dei quali possa assumere solo due stati possibili;
- i valori 0 e 1 possono anche essere interpretati come *valori di verità*, identificando per esempio 0 con *falso* e 1 con *vero*; in questo modo, data una qualunque proposizione (=affermazione), essa avrà valore logico 0 oppure 1. Ad esempio, la proposizione “ $7 > 5$ ” vale 1 (è vera), mentre la proposizione “ $7 > 13$ ” vale 0 (è falsa). La proposizione “ $x^2 > 2x - 1$ ” è una proposizione parametrica (cioè, dipende da un parametro x), e a seconda dei valori del parametro può essere vera (1) o falsa (0); in particolare, è vera se e solo se x è tale per cui $x^2 - 2x + 1 > 0$, cioè $(x - 1)^2 > 0$ il che succede se e solo se $x \neq 1$: quindi, l'affermazione di cui sopra vale 1 (vero) se $x \neq 1$, e vale 0 se $x = 1$.

Qualunque sia l'interpretazione che diamo degli elementi dell'insieme B , in questa sezione mostreremo come tali valori possono essere manipolati e faremo

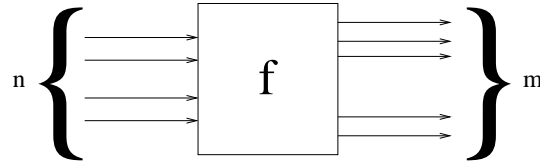


Figura 3: Una funzione booleana generale con fan-in n e fan-out m .

vedere alcune proprietà importanti delle funzioni booleane (cioè, delle funzioni che hanno B come dominio), che trovano applicazioni sia nella costruzione dei circuiti digitali sia nella pratica della programmazione quando si abbia a che fare con condizioni logiche.

3.1 Funzioni booleane e tabelle di verità

Chiamiamo *funzione booleana generale* una qualunque funzione $f : B^n \rightarrow B^m$; potete pensare a una funzione di questo tipo come a una “scatola nera” che riceve n segnali di input (ciascuno dei quali può essere 0 oppure 1) e emette m segnali di output (ciascuno dei quali, ancora una volta, può essere 0 oppure 1), come schematizzato in Figura 3: n viene chiamato *fan-in* o *arietà* della funzione f , mentre m viene chiamato *fan-out*.

Detto in altri termini, la funzione f associa, a ciascun vettore di n zeri o uni, tipo

$$\underbrace{(0, 1, 0, 0, \dots, 1, 0, 1, 1)}_n$$

un vettore di m zeri o uni, tipo

$$\underbrace{(1, 0, 0, 1, \dots, 0, 0, 1, 0)}_m.$$

La cosa avviene in modo funzionale (cioè, il vettore di output dipende esclusivamente dal vettore di input: a fronte dello stesso input la funzione produrrà lo stesso output). Considerare funzioni generali, cioè funzioni con fan-out $m > 1$, può essere scomodo, ed è inutile in quanto ogni funzione con fan-out m può essere pensata come m *funzioni booleane (semplici)* (cioè, con fan-out 1), come indicato in Figura 4: lo stesso input viene fornito alle m funzioni booleane $f_1, \dots, f_m : B^n \rightarrow B$, ognuna delle quali produce un singolo bit.

Una funzione booleana $f : B^n \rightarrow B$ può essere descritta completamente mediante la sua *tabella di verità*: si tratta di una tabella che riporta sul lato sinistro tutte le possibili n -ple di bit (in un ordine arbitrario), e sul lato destro il valore (0 o 1) della funzione in corrispondenza di quello specifico input.

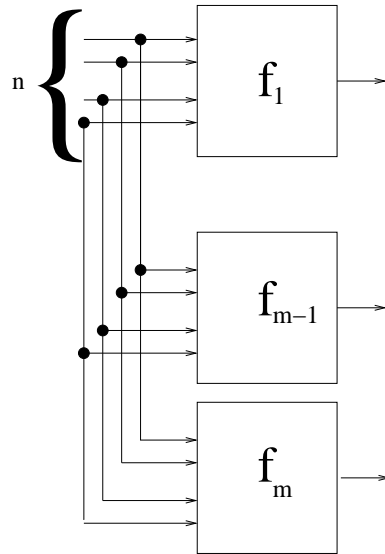


Figura 4: Una funzione booleana generale con fan-in n e fan-out m vista come m funzioni booleane semplici con fan-in n .

Questa tabella avrà 2^n righe (poiché per ognuno degli n argomenti esistono due valori possibili); sebbene l'ordine in cui compaiono le righe della tabella sia irrilevante, di solito gli input si dispongono nell'ordine dei valori decimali corrispondenti (cioè, si mette per primo il vettore $(0, 0, \dots, 0)$ che corrisponde alla rappresentazione del valore 0, e per ultimo il vettore $(1, 1, \dots, 1)$ che corrisponde alla rappresentazione del valore $2^n - 1$). La Tabella 2 mostra una tabella di verità per una funzione con arietà 3.

La tabella di verità descrive completamente la funzione, perché indica quale output la funzione deve produrre a fronte di ogni possibile input. Inoltre,

x	y	z	$f(x, y, z)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Tabella 2: Una tabella di verità per una funzione di arietà 3.

x	$f_1(x)$	x	$f_2(x)$	x	$f_3(x)$	x	$f_4(x)$
0	0	0	1	0	0	0	1
1	0	1	1	1	1	1	0

Tabella 3: Le quattro funzioni booleane di arietà 1.

tabelle di verità diverse descrivono funzioni diverse¹⁰.

Poiché una funzione $f : B^n \rightarrow B$ è completamente descritta dalla sua tabella di verità che ha 2^n righe, e poiché per ogni riga possiamo decidere se la funzione deve avere output 0 oppure 1, esistono 2^{2^n} diverse funzioni booleane di arietà n .

Ad esempio, quindi, esistono $2^{2^1} = 4$ funzioni di arietà 1, $2^{2^2} = 2^4 = 16$ funzioni di arietà 2, $2^{2^3} = 2^8 = 256$ funzioni di arietà 3 ecc. A titolo di esempio, la Tabella 3 mostra (le tabelle di verità di) tutte e quattro le funzioni di arietà 1. Come si vede, esistono due funzioni costanti (indicate con f_1 e con f_2), cioè il cui output è indipendente dall'input (ed è uguale a 0 nel caso di f_1 , e uguale a 1 nel caso di f_2); la funzione f_3 viene chiamata *identità* perché l'output coincide con l'input; infine, f_4 è una importante funzione chiamata *negazione* (o NOT), di cui avremo modo di parlare nella prossima sezione.

3.2 Un esempio pratico

Per suggerire fin d'ora le possibili applicazioni pratiche delle funzioni booleane, consideriamo il seguente problema. La Figura 5 mostra un tipico display a cristalli liquidi: si tratta di un oggetto costituito da sette barre di cristalli liquidi. Ogni barra può essere illuminata oppure no: ogni volta che si vuole visualizzare una specifica cifra si devono scegliere le barre da illuminare.

In pratica, esiste un circuito elettronico che, presa in ingresso la cifra da visualizzare, decide quali barre accendere e quali tenere spente. Più precisamente, per ognuna delle sette barre c'è un circuito che, ricevendo in ingresso la cifra da visualizzare, decide se la barra va accesa oppure no. La cifra viene fornita in input come un numero binario di quattro bit: servono quattro bit perché con tre bit si hanno solo $2^3 = 8$ combinazioni, mentre con quattro se ne hanno $2^4 = 16$ e le cifre possibili sono dieci. Quindi, gli input possibili del circuito sono $(0, 0, 0, 0)$ (corrispondente alla cifra 0), $(0, 0, 0, 1)$ (corrisponden-

¹⁰Si noti che qui quando diciamo “diverse” intendiamo che per almeno un input i due output prodotti devono essere diversi; ovvero che, se l'ordine degli input nelle due tabelle è lo stesso, l'ultima colonna delle due tabelle è diversa. Due tabelle che differiscano solo per l'ordine delle righe *non sono diverse*.



Figura 5: Un display a cristalli liquidi; la barra evidenziata è quella considerata in questo esempio.

te alla cifra 1), eccetera, fino a $(1, 0, 0, 1)$ (corrispondente alla cifra 9); gli altri 6 input sono impossibili, e quindi l'output prodotto dal circuito è irrilevante.

Consideriamo, come esempio, la barra evidenziata nella Figura 5. La barra va accesa solo in corrispondenza delle cifre 0 (corrispondente all'input $(0, 0, 0, 0)$), 2 (corrispondente all'input $(0, 0, 1, 0)$), 3 (corrispondente all'input $(0, 0, 1, 1)$), 5 (corrispondente all'input $(0, 1, 0, 1)$), 6 (corrispondente all'input $(0, 1, 1, 0)$), 7 (corrispondente all'input $(0, 1, 1, 1)$), 8 (corrispondente all'input $(1, 0, 0, 0)$) e 9 (corrispondente all'input $(1, 0, 0, 1)$). Quindi il circuito che decide se accendere oppure no la barra deve avere la seguente tabella di verità (nelle righe in cui l'output indicato è ? la funzione può assumere qualunque valore):

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	0	?
1	0	1	1	?
1	1	0	0	?
1	1	0	1	?
1	1	1	0	?
1	1	1	1	?

NOT $x, \bar{x}, !x$ x OR $y, x + y, x||y$ x AND $y, x \cdot y, xy, x\&y$

x	$\neg x$
0	1
1	0

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

Tabella 4: Le funzioni NOT, OR e AND.

3.3 Funzioni di base ed espressioni logiche

Come abbiamo visto, ogni funzione booleana si può descrivere mediante la sua tabella di verità. Un modo alternativo per ottenere lo stesso risultato consiste nell'usare le *espressioni booleane* (o espressioni logiche). Normalmente, si procede come segue: si fissa un insieme \mathcal{F} di funzioni booleane (di diverse arietà), dette funzioni di base; quindi, si combinano queste funzioni in espressioni ottenendo così nuove funzioni.

Le funzioni più usate a questo scopo sono le funzioni AND, OR e NOT, le cui tabelle di verità sono indicate in Tabella 4. Eccone una breve descrizione informale:

- La funzione *NOT* (indicata in queste note come $\neg x$, ma che si può anche trovare indicata come NOT x oppure $!x$ oppure \bar{x}) è anche detta *negazione*, ed è una funzione unaria (cioè, con un solo input) che inverte il valore dell'input: cioè, produce 1 se l'input è 0, e 0 se l'input è 1. La funzione NOT ha il significato logico dell'avverbio “non”: se A è una proposizione e $\neg A$ è la proposizione “non A ”, allora se A è vero $\neg A$ è falso, e viceversa. Ad esempio, se $A =$ “oggi piove”, allora $\neg A =$ “oggi non piove” e chiaramente se A è vera, $\neg A$ è falsa, mentre se A è falsa, $\neg A$ è vera.
- La funzione *AND* (indicata in queste note come $x \wedge y$, ma che si può anche trovare indicata come x AND y oppure $x \cdot y$ oppure xy oppure $x\&y$) è anche detta *congiunzione*, ed è una funzione binaria (cioè, con due input) che vale sempre falso, tranne nel caso che *entrambi* gli input siano veri. La funzione AND ha il significato logico della congiunzione “e”: se A e B sono due proposizioni e $A \wedge B$ indica la proposizione “ A e B ”, allora perché “ A e B ” sia vero occorre che sia A che B siano vere. In altri termini, se Tizio vi dice “oggi piove e la temperatura è superiore ai 30 gradi” perché Tizio non vi abbia mentito occorre che abbia detto

la verità su entrambe le affermazioni (cioè, occorre che effettivamente oggi piova e che la temperatura sia superiore ai 30 gradi); se anche una sola delle due affermazioni fosse falsa, l'intera affermazione di Tizio lo sarebbe, e quindi Tizio avrebbe mentito.

- La funzione *OR* (indicata in queste note come $x \vee y$, ma che si può anche trovare indicata come $x \text{ OR } y$ oppure $x + y$ oppure $x || y$) è anche detta *disgiunzione*, ed è una funzione binaria (cioè, con due input) che vale sempre vero, tranne nel caso che *entrambi* gli input siano falsi. La funzione OR ha il significato logico della congiunzione “oppure” (non esclusivo): se A e B sono due proposizioni e $A \vee B$ indica la proposizione “ A oppure B ”, allora perché “ A oppure B ” sia vero basta che sia vera A o B , non è necessario che lo siano entrambe. In altri termini, se Tizio vi dice “oggi piove oppure la temperatura è superiore ai 30 gradi” perché Tizio vi abbia mentito occorre che abbia mentito su entrambe le affermazioni (cioè, occorre che oggi non piova e che la temperatura non sia superiore ai 30 gradi); se anche una sola delle due affermazioni fosse vera, l'intera affermazione di Tizio lo sarebbe, e quindi Tizio avrebbe complessivamente detto la verità.

Ecco alcuni esempi di espressioni logiche che coinvolgono le funzioni AND, OR e not¹¹:

$$\begin{aligned} &(x \vee y) \wedge \neg z \\ &y \wedge (x \vee x) \\ &(\neg x \vee y) \wedge (x \wedge z) \end{aligned}$$

Ogni espressione può essere pensata come una funzione: se un'espressione contiene n variabili, per ognuna delle 2^n possibili scelte dei valori delle variabili l'espressione assumerà un certo valore. Ad esempio, l'espressione $(x \vee y) \wedge \neg z$ quando $x = 1$, $y = 0$ e $z = 0$ vale $(1 \vee 0) \wedge \neg 0 = (1 \vee 0) \wedge 1 = 1 \wedge 1 = 1$. Un'espressione può essere dunque un modo “compatto” per descrivere una funzione.

Tabella di verità di un'espressione. Un problema che si pone abbastanza di frequente è, data un'espressione logica, di calcolarne la tabella di verità. In pratica, questo significa determinare il valore dell'espressione per ogni possibile combinazione dei valori delle variabili. La tabella di verità

¹¹Nella scrittura, assumeremo sempre che \neg abbia priorità maggiore di \vee e \wedge ; scriveremo, ad esempio $x \vee \neg z$ per intendere $x \vee (\neg z)$.

rappresenta la descrizione della funzione che l'espressione rappresenta. Il modo più semplice per calcolare la tabella di verità di un'espressione consiste nello scomporre l'espressione che ci interessa in sottoespressioni via via più semplici, e nel riempire una tabella che contenga i valori intermedi di queste espressioni.

Considerate ad esempio l'espressione $(x \vee y) \wedge (\neg x \vee z)$; si tratta di un'espressione che contiene tre variabili (x , y e z) e quindi la sua tabella di verità avrà $2^3 = 8$ righe. Come si vede, l'espressione è costituita da due sottoespressioni più semplici: $x \vee y$ e $\neg x \vee z$; la prima sottoespressione è calcolabile direttamente a partire dai valori di x e y , mentre la seconda è a sua volta scomponibile nelle sottoespressioni $\neg x$ e z .

La seguente tabella rappresenta il processo: sul lato sinistro troviamo le 2^8 combinazioni dei valori di input (come avviene in qualunque tabella di verità). Nella parte intermedia, ci sono le sottoespressioni via via più complesse, fino a giungere, nell'ultima colonna, al valore dell'espressione.

x	y	z	$x \vee y$	\bar{x}	$\bar{x} \vee z$	$(x \vee y) \wedge (\bar{x} \vee z)$
0	0	0	0	1	1	0
0	0	1	0	1	1	0
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	0	0
1	0	1	1	0	1	1
1	1	0	1	0	0	0
1	1	1	1	0	1	1

Espressioni equivalenti, tautologie e contraddizioni. Due espressioni sono dette *equivalenti* se rappresentano la stessa funzione; per esempio, se diciamo che $y \vee (x \wedge (x \vee y))$ e $x \vee y$ sono equivalenti intendiamo che definiscono la stessa funzione, cioè che hanno la stessa tabella di verità: quest'ultimo è in effetti il modo più diretto per dimostrare che due espressioni sono equivalenti. Quando scriviamo

$$y \vee (x \wedge (x \vee y)) = x \vee y$$

intendiamo dire con il simbolo $=$ che le due espressioni sono equivalenti¹².

Un'espressione è detta essere una *tautologia* se vale sempre 1, indipendentemente dal valore delle variabili; è detta una *contraddizione* se vale sempre 0, indipendentemente dal valore delle variabili.

¹²Notate che è possibile che due espressioni con un numero *diverso* di variabili siano equivalenti; ad esempio, è facile verificare che $x \vee (x \wedge y) = x$ il che implica, ovviamente, che i valori dell'espressione di sinistra sono indipendenti da y .

3.4 Proprietà algebriche

Le funzioni AND, OR e NOT soddisfano una serie di proprietà; il seguente teorema ne riassume alcune sottoforma di equivalenza fra espressioni.

Teorema 10. *Valgono le seguenti equivalenze fra espressioni logiche:*

- proprietà associativa: $(x \vee y) \vee z = x \vee (y \vee z)$ e $(x \wedge y) \wedge z = x \wedge (y \wedge z)$
- proprietà commutativa: $x \vee y = y \vee x$ e $x \wedge y = y \wedge x$
- elementi neutri: $x \vee 0 = x$ e $x \wedge 1 = x$
- proprietà di assorbimento: $x \vee (x \wedge y) = x$ e $x \wedge (x \vee y) = x$
- proprietà di annullamento: $x \vee 1 = 1$ e $x \wedge 0 = 0$
- proprietà distributiva: $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ e $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
- complementazione: $x \vee \neg x = 1$ e $x \wedge \neg x = 0$
- doppia negazione: $\neg(\neg x) = x$
- idempotenza: $x \wedge x = x$ e $x \vee x = x$
- leggi di De Morgan: $\neg(x \vee y) = \neg x \wedge \neg y$ e $\neg(x \wedge y) = \neg x \vee \neg y$.

Dimostrazione. Tutte queste proprietà si possono dimostrare facilmente esibendo le tabelle di verità dei due membri (destro e sinistro) delle uguaglianze e verificando che le tabelle ottenute sono uguali. Ci limiteremo a mostrare come procedere per verificare la prima delle due leggi di De Morgan. La tabella di verità di $\neg(x \vee y)$ è:

x	y	$x \vee y$	$\neg(x \vee y)$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

La tabella di verità di $\neg x \wedge \neg y$ è:

x	y	$\neg x$	$\neg y$	$\neg x \wedge \neg y$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Come vedete le due tabelle sono uguali. □

Alcune osservazioni:

- poiché vale la proprietà associativa, nello scrivere espressioni che coinvolgono solo \vee o solo \wedge (ad esempio, $x \vee y \vee z$) si possono omettere le parentesi;
- le parentesi servono, invece, nelle espressioni in cui compaiano \vee e \wedge mescolati: ad esempio $(x \vee y) \wedge z$ è diversa da $x \vee (y \wedge z)$; normalmente, si assume che \wedge abbia priorità maggiore, e quindi $x \vee y \wedge z$ è da interpretarsi come $x \vee (y \wedge z)$, ma noi eviteremo di fare questa assunzione e metteremo sempre le parentesi in casi del genere;
- una volta che si sia dimostrata un'equivalenza fra due espressioni, la si può usare per dimostrarne altre. Ad esempio, per dimostrare che $y \vee (x \wedge \neg(x \vee (x \wedge y))) = y$ si può procedere come segue:

$$y \vee (x \wedge \neg(x \vee (x \wedge y))) = \quad (\text{assorbimento})$$

$$y \vee (x \wedge \neg x) = \quad (\text{complementazione})$$

$$y \vee 0 = \quad (\text{elemento neutro})$$

$$y;$$

- qualunque struttura algebrica che soddisfi la proprietà associativa e abbia l'elemento neutro viene chiamata *monoide*, abeliano o commutativo se soddisfa anche la proprietà commutativa. Una struttura con due operazioni che sia un monoide abeliano rispetto ad entrambe e per cui valga anche l'assorbimento si chiama *reticolo*, e *reticolo distributivo* se vale anche la proprietà distributiva; un reticolo distributivo per cui valga la complementazione si chiama *algebra di Boole*.

3.5 Completezza

Come abbiamo visto, ogni espressione logica denota una funzione, ma è ragionevole chiedersi se valga il viceversa; se cioè ogni funzione booleana sia denotabile attraverso una qualche espressione. Questo dipende dalla scelta della base di funzioni \mathcal{F} ; l'importanza della base $\{\vee, \wedge, \neg\}$ sta nel fatto che questa base è completa, cioè permette di denotare ogni funzione:

Teorema 11. *Ogni funzione booleana si può denotare mediante un'espressione ottenuta componendo solo \neg , \vee e \wedge ; ovvero, $\{\vee, \wedge, \neg\}$ è una base completa.*

Dimostrazione. Sia $f : B^n \rightarrow B$ una qualunque funzione booleana di arietà n , e consideriamo un qualche $(a_1, \dots, a_n) \in B^n$. A questo vettore di zeri e uni associamo una formula che chiameremo *mintermine del vettore* (a_1, \dots, a_n) ottenuta come segue: la formula è un AND in cui compaiono le variabili x_1, \dots, x_n , e ciascuna variabile x_i compare in forma normale o negata a seconda che $a_i = 1$ o $a_i = 0$. Ad esempio, al vettore $(0, 1, 1, 0, 1)$ assoceremo il mintermine $\neg x_1 \wedge x_2 \wedge x_3 \wedge \neg x_4 \wedge x_5$. È facile vedere che il mintermine associato al vettore (a_1, \dots, a_n) è un'espressione che è sempre falsa *tranne* che sull'input (a_1, \dots, a_n) .

Ora supponete che la tabella di verità di f contenga esattamente k righe su cui f vale 1, e considerate i k mintermini corrispondenti. L'OR di questi mintermini sarà una espressione che rappresenta f . \square

La dimostrazione di questo teorema è *costruttiva*, nel senso che non si limita a mostrarci che per ogni funzione c'è un'espressione che la rappresenta, ma ci dice anche come ottenere un'espressione che rappresenti la funzione.

Per farlo vedere, torniamo all'esempio del display, presentato nella Sezione 3.2. Avevamo detto che il circuito che faceva funzionare la barretta doveva realizzare una funzione con la seguente tabella di verità:

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	0	(0)
1	0	1	1	(0)
1	1	0	0	(0)
1	1	0	1	(0)
1	1	1	0	(0)
1	1	1	1	(0)

Gli zeri fra parentesi sono valori arbitrari, ma conviene assumere che si tratti di zeri. La tabella ha sette righe su cui vale 1, e l'OR dei mintermini

corrispondenti è:

$$\begin{aligned} & (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3} \wedge \overline{x_4}) \vee (\overline{x_1} \wedge \overline{x_2} \wedge x_3 \wedge \overline{x_4}) \vee (\overline{x_1} \wedge \overline{x_2} \wedge x_3 \wedge x_4) \vee \\ & (\overline{x_1} \wedge x_2 \wedge \overline{x_3} \wedge x_4) \vee (\overline{x_1} \wedge x_2 \wedge x_3 \wedge \overline{x_4}) \vee \\ & (\overline{x_1} \wedge x_2 \wedge x_3 \wedge x_4) \vee (x_1 \wedge \overline{x_2} \wedge \overline{x_3} \wedge \overline{x_4}). \quad (1) \end{aligned}$$

Potete verificare che in effetti questa espressione ha la tabella di verità desiderata. Naturalmente, non è detto che questa sia *la più semplice espressione che rappresenta la funzione desiderata* (ammesso di precisare che cosa si intenda esattamente per “semplice”).

Un ultimo corollario: usando le leggi di De Morgan, è sempre possibile sostituire AND e OR al prezzo di aggiungere qualche NOT in più. Quindi, vale il seguente

Corollario 12. *Anche $\{\vee, \neg\}$ e $\{\wedge, \neg\}$ sono basi complete.*

3.5.1 Alcune funzioni notevoli

Il fatto che le basi presentate siano complete non significa che siano *le sole* basi complete, né che siano le più utilizzate. Elenchiamo qui una serie di altre funzioni booleane binarie interessanti, alcune delle quali costituiscono (da sole o insieme ad altre) basi complete, e che si possono ovviamente esprimere come funzione di AND/OR/NOT.

- NAND (indicato come $\overline{\wedge}$): $x \overline{\wedge} y = \overline{x \wedge y}$
- NOR (indicato come $\overline{\vee}$): $x \overline{\vee} y = \overline{x \vee y}$
- XOR o “or esclusivo” (indicato come \oplus): $x \oplus y = (\overline{x} \wedge y) \vee (x \wedge \overline{y})$
- IMPLICA (indicato come $x \implies y$ o $x \supset y$): $x \implies y = \overline{x} \vee (x \wedge y)$
- SSE (indicato come $x \Leftrightarrow y$ o $x == y$): $x \Leftrightarrow y = (x \implies y) \wedge (y \implies x)$.

3.6 Circuiti booleani

Come abbiamo già osservato, si può pensare a una funzione booleana $f : B^n \rightarrow B$ come a una “scatola nera” che ha n ingressi e una uscita: ingressi e uscite sono costituiti da valori booleani, rappresentati in qualche modo. Se immaginate che i valori booleani siano rappresentati ad esempio da due livelli di tensione, potete pensare alla scatola nera come a un vero e proprio circuito elettronico, in cui gli input e gli output sono fisicamente costituiti

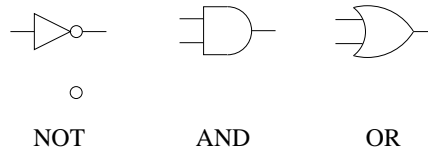


Figura 6: Rappresentazione grafica delle porte NOT, AND e OR.

da fili elettrici che conducono i valori booleani. Alla luce di questa visione “circuitale” della logica booleana, il teorema di completezza ha la seguente interpretazione: se è possibile realizzare dei circuiti elettronici in grado di implementare le funzioni AND, OR e NOT, allora componendo questi circuiti si può realizzare qualunque funzione booleana attraverso un circuito.

Le funzioni AND, OR e NOT sono, in effetti, realizzabili elettronicamente, e quindi, componendo opportunamente queste porte, si possono realizzare circuiti booleani per funzioni arbitrarie. È anzi in questo modo che si costruiscono le CPU.

La Figura 6 mostra il modo con cui si indicano, graficamente, le porte NOT, AND e OR, cioè i componenti che realizzano le corrispondenti funzioni: osservate che la porta NOT si può indicare con un triangolo oppure, talvolta, con un cerchio vuoto (è questa la rappresentazione preferita quando il NOT si trova all’ingresso o all’uscita di una porta AND/OR).

A titolo di esempio, in Figura 7 mostriamo il circuito che implementa la formula (1) che realizza la funzione booleana dell’esempio di Sezione 3.2. Le quattro linee verticali, etichettate con x_1, \dots, x_4 rappresentano gli input del circuito, che contengono la codifica binaria della cifra da visualizzare. Tali input vengono distribuiti a varie porte AND che rappresentano i sette mintermini della formula (gli ingressi sono opportunamente negati; ad esempio, il primo mintermine contiene tutte e quattro le variabili in forma negata, quindi tutti gli ingressi sono passati attraverso un NOT). Le uscite delle sette porte AND sono passate alla porta OR che fornisce il risultato.

Notate che in questo circuito, come nei successivi, stiamo assumendo di avere porte AND/OR con fan-in arbitrario: nella realtà, se si dovesse disporre di porte AND/OR con un fan-in specifico (ad esempio, due) ogni singola porta AND/OR rappresentata nel circuito dovrebbe essere sostituita con un insieme di porte AND/OR, usando la proprietà di associatività.

3.7 Realizzazione di un sommatore

Come ultimo esempio di circuito, proviamo a realizzare un sommatore. Un *sommatore (adder) a k bit* è un circuito che prende in ingresso due numeri binari di k cifre ciascuno (ed ha, quindi, $2k$ input) e produce in uscita la loro

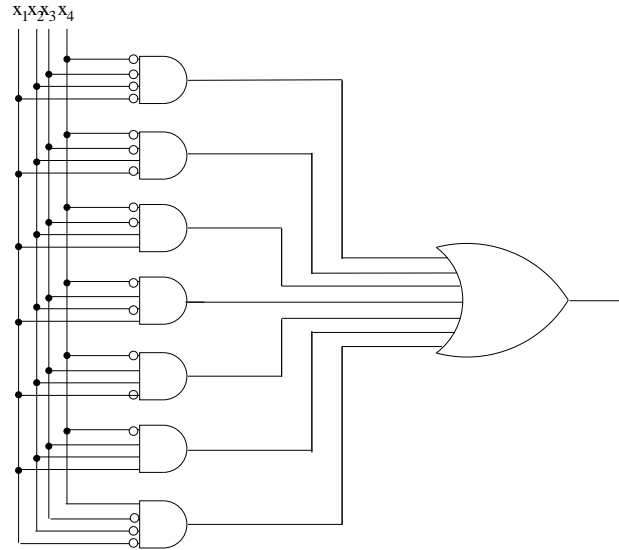


Figura 7: Il circuito per l'esempio del display a cristalli liquidi, (sezione 3.2).

somma, sempre come numero binario di k cifre, fornendo anche l'eventuale riporto (ha, quindi, $k + 1$ output). Notate che potremo usare questo circuito come elemento di una CPU che somma valori naturali oppure interi in complemento a due (si riveda la sezione in cui si è discusso come la somma in complemento a due sia equivalente alla somma di naturali).

3.7.1 Semisommatore a 3 bit

Il primo passo per la realizzazione del circuito consiste nella costruzione di un circuito più semplice che chiameremo *semisommatore* (*half-adder*). Un semisommatore è un circuito con tre ingressi che vengono interpretati come tre numeri (0 oppure 1): il semisommatore somma i tre valori in ingresso, emettendo la somma (un numero binario di due cifre, in quanto è compreso fra 0 e 3) su due linee distinte che chiameremo s (somma) e r (riporto).

Cominciamo a stabilire i valori di s e r in funzione degli input:

x_1	x_2	x_3	$s(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

x_1	x_2	x_3	$r(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Una formula per r . Ricaviamo in primo luogo una formula per r ; usando i mintermini:

$$r(x_1, x_2, x_3) = (\overline{x_1} \wedge x_2 \wedge x_3) \vee (x_1 \wedge \overline{x_2} \wedge x_3) \vee (x_1 \wedge x_2 \wedge \overline{x_3}) \vee (x_1 \wedge x_2 \wedge x_3).$$

Tentiamo ora di semplificare la formula usando le proprietà note. Ora applichiamo la proprietà commutativa (che ci permette di cambiare l'ordine dei termini) e di idempotenza (che ci consente di ripetere più volte lo stesso termine) ottenendo la formula equivalente:

$$\begin{aligned} r(x_1, x_2, x_3) &= (\overline{x_1} \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3) \vee \\ &\quad (x_1 \wedge \overline{x_2} \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3) \vee \\ &\quad (x_1 \wedge x_2 \wedge \overline{x_3}) \vee (x_1 \wedge x_2 \wedge x_3). \end{aligned}$$

Applicando tre volte la proprietà distributiva e riordinando di nuovo i termini, si ottiene:

$$\begin{aligned} r(x_1, x_2, x_3) &= (x_1 \wedge x_2 \wedge (x_3 \vee \overline{x_3})) \vee \\ &\quad (x_1 \wedge x_3 \wedge (x_2 \vee \overline{x_2})) \vee \\ &\quad (x_2 \wedge x_3 \wedge (x_1 \vee \overline{x_1})) \vee \end{aligned}$$

Se ora applichiamo la proprietà di complementazione (secondo la quale, ad esempio $x_1 \vee \overline{x_1} = 1$) e la proprietà degli elementi neutri, otteniamo:

$$r(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3).$$

Una formula per s . Sempre partendo dai mintermini, abbiamo:

$$s(x_1, x_2, x_3) = (\overline{x_1} \wedge \overline{x_2} \wedge x_3) \vee (\overline{x_1} \wedge x_2 \wedge \overline{x_3}) \vee (x_1 \wedge \overline{x_2} \wedge \overline{x_3}) \vee (x_1 \wedge x_2 \wedge x_3).$$

Ora, usando la proprietà distributiva e quella commutativa, si ottiene la formula (leggermente più semplice)

$$\begin{aligned} s(x_1, x_2, x_3) &= (x_1 \wedge ((x_2 \wedge x_3) \vee (\overline{x_2} \wedge \overline{x_3}))) \vee \\ &\quad (\overline{x_1} \wedge ((\overline{x_2} \wedge x_3) \vee (x_2 \wedge \overline{x_3}))). \end{aligned}$$

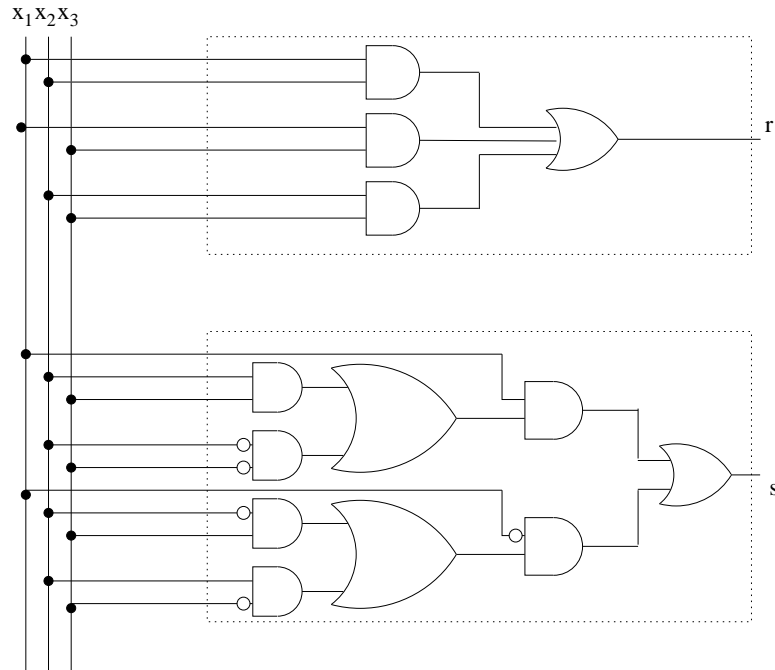


Figura 8: Il circuito del semisommatore a 3 bit.

Il circuito. Il circuito che implementa le due formule per r e s ricavate è visualizzato in Figura 8: il blocco superiore (distinto dal tratteggio) è quello relativo a r , mentre il blocco inferiore è quello relativo a s .

3.7.2 Il sommatore

Ora, la realizzazione del sommatore a k bit si può ottenere componendo molti semisommatori a 3 bit. In particolare, per sommare due numeri di k bit, diciamo $x_{k-1} \dots x_0$ e $y_{k-1} \dots y_0$ ottenendo una somma $z_{k-1} \dots z_0$ si procede come segue:

- si sommano i bit meno significativi x_0 e y_0 ; il risultato ottenuto è z_0 ;
- si sommano i bit x_1 e y_1 , e si aggiunge anche il riporto r_0 ottenuto al passo precedente; il risultato ottenuto è z_1 ;
- si sommano i bit x_2 e y_2 , e si aggiunge anche il riporto r_0 ottenuto al passo precedente; il risultato ottenuto è z_2 ;

e così via. È evidente che ogni passo si può realizzare usando un semisommatore a 3 bit (la prima somma fa eccezione, poiché sommiamo solo due bit, ma possiamo prendere come terzo ingresso il valore costante 0). Il circuito

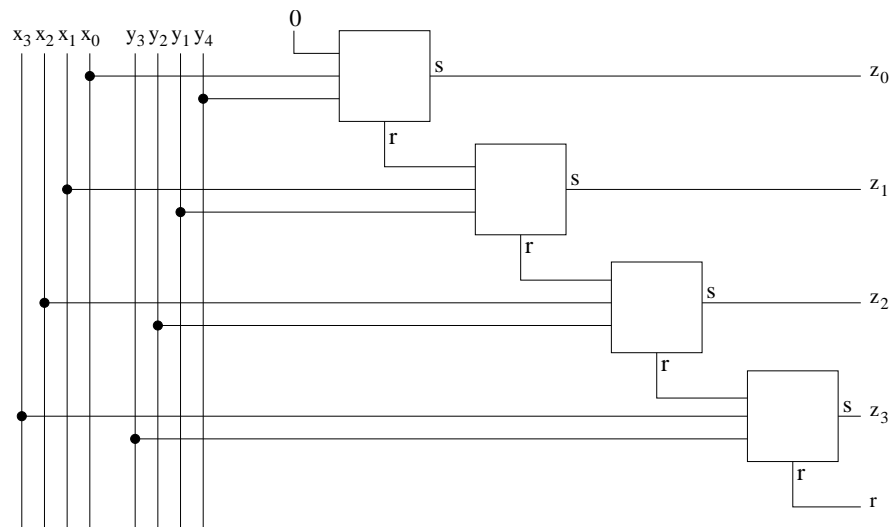


Figura 9: Il circuito del sommatore a k bit (in questo esempio $k = 4$); gli input sono $x_{k-1} \dots x_0$ e $y_{k-1} \dots y_0$, l'output è $z_{k-1} \dots z_0$, e il riporto finale è r .

ottenuto è rappresentato in Figura 9: ogni singolo rettangolo nero rappresenta un'istanza di un semisommatore a 3 bit; le due uscite sono etichettate con s e r per distinguerle.