Alessandro Simonetta
Marco Sillano
Daniele Perna

INFORMAZIONE AUTOMATICA e JAVA

Compendio *di* Informatica *e di* Programmazione

Edizioni Kappa

INFORMAZIONE AUTOMATICA E JAVA

INFORMAZIONE AUTOMATICA E JAVA è dedicato a chi si avvicina per la prima volta al mondo della programmazione e raccoglie in un'ampia visione unificante le tecnologie più affermate (Java, JDBC, UML, HTML, XML, XSLT, SQL). Gli argomenti sono presentati secondo una nuova concezione didattica ed editoriale incentrata sulla percezione visiva e sulla tecnica del learn-by-example che facilita la rapida assimilazione dei concetti, evidenziando la sostanziale convergenza delle discipline informatiche.

ARGOMENTI CHIAVE
- Anatomia di un calcolatore: hardware e software
- Il funzionamento del microprocessore
- ABC della programmazione e Java
- Tecniche di progettazione Object-Oriented
- Java e i database (JDBC e SQL)
- Il linguaggio di modellazione UML e il ciclo di vita del software
- Reti di calcolatori e protocolli
- I linguaggi per il web (HTML, XML e XSLT)

ALESSANDRO SIMONETTA è docente a contratto dei corsi di *"Fondamenti di informatica con elementi di programmazione"* e di "Tecniche Informatiche" presso la Facoltà di Psicologia1 dell'Università "La Sapienza" di Roma. Ha svolto attività di ricerca nel settore dell'intelligenza artificiale e si è occupato della progettazione software di ambienti di supervisione e controllo delle reti di telecomunicazioni. Attualmente è professionista informatico della Consulenza per l'Innovazione Tecnologica dell'INAIL.

MARCO SILLANO ha oltre 30 anni di esperienza di divulgazione didattica, come docente di elettronica e di informatica. Negli ultimi anni si è occupato di web e della progettazione dei sistemi di e-commerce. Ha curato una rubrica fissa sul mensile di settore *MCmicrocomputer*, ha scritto numerosi libri di didattica e di informatica per vari Editori (O.S., Gremese, S.E.I.).

DANIELE PERNA è docente a contratto del corso di *"Informatica"* per il Dipartimento di Ingegneria Chimica presso la Facoltà di Ingegneria dell'Università "La Sapienza" di Roma. Attualmente è responsabile di area nel settore web per la Direzione Centrale Servizi Informativi e Telecomunicazioni dell'INAIL.

www.informazioneautomaticaejava.info

Edizioni Kappa

ISBN 88-7890-659-X
9 788878 906594

€ 35,00

INDICE

PREMESSA

Il presente volume è dedicato ai lettori che si avvicinano per la prima volta al mondo della programmazione senza alcuna nozione riguardo i fondamenti dell'informatica. L'ambizioso progetto degli autori è quello di condurre il lettore neofita in un viaggio fantastico che a partire dai concetti di base dell'informatica lo accompagni fino alla progettazione di sistemi complessi con i consolidati modelli dell'ingegneria del software.

In un mondo sempre più orientato verso l'innovazione tecnologica regolato dall'equazione secondo cui massimizzare l'informazione automatica corrisponde ad incrementare il settore di business minimizzando costi e tempi, l'aggiornamento professionale delle risorse assume un ruolo chiave. Nasce così l'esigenza di un testo completo che affronti le tecnologie più affermate (Java™, JDBC™, UML, HTML, XML, XSLT, SQL) in una visione metodologica unificata che evidenzi la sostanziale convergenza di settori tradizionalmente separati. Gli argomenti sono presentati secondo una nuova concezione didattica ed editoriale incentrata sulla percezione visiva e sulla tecnica del *learn-by-example* che deriva dalla vasta esperienza degli autori nel settore.

Anche se è stato pensato originariamente per gli studenti di Psicologia del Corso di Laurea triennale e Laurea Specialistica che devono sostenere esami di informatica, questo testo può essere un valido ausilio per gli studenti di altre facoltà scientifiche o per gli studenti di Istituti Tecnici e Professionali che si trovano di fronte al problema di acquisire nozioni di base della programmazione senza un adeguato livello di conoscenza iniziale.

Gli autori si scusano anticipatamente per eventuali errori e/o sviste presenti nel testo e ringraziano tutti coloro i quali vorranno segnalarli attraverso il canale email.

Tutti i marchi citati nel testo sono di proprietà dei rispettivi proprietari.

INTRODUZIONE

Il capitolo 1 introduce le nozioni di base dei sistemi digitali ed analogici, i sistemi di numerazione, e il dominio dell'algebra binaria ove è possibile scoprire tecniche di sintesi di algoritmi attraverso soluzioni innovative relative a problemi concreti (sistemi di monitoraggio ambientale, sistemi di allarme,…).

Il capitolo 2 si occupa delle architetture hardware con particolare riferimento al modello di J. Von Neumann e agli attuali componenti elettronici (CPU, memoria, bus, etc.). Nel capitolo si descrive il funzionamento degli elaboratori con le problematiche relative all'architettura interna della CPU, all'elettronica del bus, alla gestione della memoria e alla gestione dei dispositivi periferici. Il primo linguaggio di programmazione, il linguaggio assembler 8086, viene introdotto a scopo esemplificativo.

Nel capitolo 3 vengono rappresentate le architetture software a cominciare dal sistema operativo fino alle applicazioni di alto livello. Il concetto di astrazione presente nei sistemi operativi riveste un ruolo chiave che verrà poi ripreso nei modelli di progettazione object-oriented (capitolo 8) e nei protocolli di rete (capitolo 11).

Il testo utilizza un modello di programmazione imperativo rendendo quindi necessaria una fase di ricognizione (capitolo 4) sui possibili linguaggi di programmazione (tassonomia dei linguaggi di programmazione). La differenza che intercorre tra compilatori ed interpreti, oltre ad avere una valenza teorica fondamentale, aiuta ad introdurre il concetto di portabilità di Java. Questo capitolo rappresenta una valida introduzione per i linguaggi di programmazione: Java, SQL e XSLT.

Il capitolo 5 introduce il lettore nel mondo della programmazione, in queste pagine vengono messe a frutto le competenze acquisite nei precedenti capitoli e viene introdotta la metodologia di I. Nassi e B. Snaidermann (DNS). I DNS risultano personalizzati secondo le esigenze degli autori per introdurre concetti fondamentali quali: strutturazione del codice, forma del codice (indentazione), e conseguentemente alta leggibilità e apprendimento.

Nonostante la metodologia risalga ad epoche passate è ancora valida didatticamente e ben si collega con i modelli UML.

Il problema tipico del programmatore neofita che non ha seguito corsi di informatica di base risiede nella incapacità di rappresentare processi risolutivi. Lo scopo del capitolo 6 è di introdurre una serie di esercizi, con modalità incrementale, per far acquisire al lettore la capacità a realizzare algoritmi risolutivi. I capitoli 5 e 6 prescindendo dal linguaggio di programmazione possono essere di ausilio per l'implementazione in qualsiasi linguaggio imperativo (Java, Delphi, C#, C++, …).

Nel capitolo 7 entra in gioco la fase di traduzione dell'algoritmo nel linguaggio di programmazione (Java): verranno mostrati in questa sede tutti gli schemi di traduzione dei blocchi introdotti nei capitoli precedenti mediante l'ausilio di un package ideato dagli autori allo scopo di renderne più semplice la traduzione. Ovviamente trovano riscontro in Java tutti gli esercizi proposti nel capitolo 6.

Nel capitolo 8 vengono illustrati i concetti fondamentali della programmazione orientata agli oggetti: l'astrazione (incapsulamento ed information hiding), l'ereditarietà e il polimorfismo. Vengono inoltre introdotti i modelli iniziali dell'UML per la gestione dell'OOA (object-oriented analysis) con esempi concreti e realizzazioni in Java.

Il tipo di dato astratto è un concetto teorico che trova applicazione negli ambienti object-oriented; lo scopo del capitolo 9 è quello di riassumere alcune implementazioni in Java di strutture di dati standard. Il capitolo fornisce la descrizione di librerie fondamentali Java: le classi per la gestione dell'Input/Output, per la gestione delle collezioni di dati e per l'accesso ai database commerciali JDBC™.

Il capitolo 10 è dedicato al linguaggio UML e al processo di produzione del software USDP: in esso sono descritte le caratteristiche salienti della metodologia con vari esempi pratici. Nel capitolo sono anche presenti delle *best-practice* relative alla realizzazione pratica dei diagrammi UML.

Nel capitolo 11 vengono introdotti i concetti relativi alle reti di calcolatori, fino a giungere alla rete internet e il WWW. Si prosegue poi con una descrizione puntuale dei linguaggi HTML, XML e XSLT, arricchita da numerosi esempi allo scopo di renderne più agevole la comprensione. Il capitolo descrive inoltre i modelli relativi alle architetture web-based.

Il capitolo 12 è dedicato ai sistemi per la gestione delle basi di dati (dbms). Nel capitolo viene presentata una metodologia per la traduzione dei Class Diagram (UML) in schemi logici relazionali. La sezione conclusiva è dedicata al linguaggio di interrogazione SQL che viene descritto seguendo un approccio funzionale.

1 LA RAPPRESENTAZIONE DELLE INFORMAZIONI

1.1 INTRODUZIONE

Questo capitolo presenta le operazioni logiche (o *algebra di Boole*) che sono alla base del funzionamento dei circuiti dei calcolatori elettronici.

Sono operazioni molto semplici, ma basilari, che si ritrovano anche nel linguaggio Java (vedi oltre capitolo 5). Il capitolo mostra inoltre come sono memorizzati vari tipi di informazioni nei computer: numeri, immagini, musica, e per far questo introduce i vari sistemi di memorizzazione importanti per l'informatico. E' anche introdotto un modello, l'*automa a stati finiti*, che verrà utilizzato a più riprese nei capitoli successivi.

1.2 I CALCOLATORI ELETTRONICI

Il mondo che ci circonda è invaso completamente dalla tecnologia, ormai quasi tutti gli strumenti che utilizziamo quotidianamente (telefono cellulare, PDA, lettore di CD/DVD/MP3, autovettura, TV, etc.) contengono delle unità elettroniche "intelligenti". Nel prossimo futuro sarà possibile interagire con gli elettrodomestici programmandoli secondo le nostre esigenze, oppure saranno essi stessi a scambiarsi informazioni per perseguire obiettivi comuni (ad esempio si potrebbe garantire che il livello totale di potenza assorbita rimanga al di sotto di un valore prefissato).

Certo può sembrare strano che un componente elettronico possa essere dotato di intelligenza, qualifica che appartiene soltanto al genere umano. In realtà questi sistemi sono stati opportunamente programmati e il loro comportamento è stato "prestabilito" dall'intelligenza di un essere umano.

Il processo di evoluzione tecnologica negli ultimi anni è stato davvero sorprendente e il genere umano è riuscito a varcare nuove frontiere soprattutto grazie all'introduzione di queste tecnologie. Si pensi ad esempio alle ultime missioni sui pianeti del sistema solare che fino ad alcuni anni fa si ritenevano irrealizzabili.

La genesi di questa evoluzione presumibilmente risiede nella capacità di realizzare dispositivi elettronici complessi in uno spazio sempre minore e, soprattutto, con costi sempre più bassi. Possiamo certamente affermare che la miniaturizzazione dei componenti elettronici e l'avvento dell'elettronica digitale (*transistor*) hanno giocato un ruolo predominante.

Sebbene la parola digitale deriva dal termine inglese *digit* che significa letteralmente "cifra", alla parola digitale dobbiamo attribuire il significato di discreto. In antitesi alla rappresentazione o codifica digitale troviamo quella analogica in cui le grandezze assumono valori all'interno di un dominio continuo.

Un sistema elettronico digitale utilizza la rappresentazione discreta delle informazioni e le codifica mediante un sistema di soli due simboli (*bit* = *Binary digIT*). Un calcolatore (o *computer*) è un dispositivo elettronico digitale che ha la caratteristica di essere programmabile: è possibile programmare il suo comportamento allo scopo di risolvere problematiche complesse.

Con il termine *hardware* si identificano le componenti fisiche di un elaboratore: il monitor, la tastiera, il contenitore, le schede, l'alimentatore, l'harddisk, etc.

Con il termine *software* si identifica tutto ciò che è frutto dell'ingegno e quindi ciò che non è tangibile: i programmi che vengono eseguiti sull'hardware. In alcune circostanze il software può essere inserito in un hardware ovvero è possibile memorizzare un programma all'interno di un componente fisico (memoria persistente). Pensiamo ad esempio ad un qualsiasi dispositivo elettronico non programmabile (direttamente): telefono cellulare, TV, lettore di CD/DVD/MP3. In generale è sempre possibile aggiornare il software all'interno del dispositivo.

1.3 L'ALGEBRA DI BOOLE

L'origine di tutto è in alcune semplici operazioni scoperte già dall'antichità analizzando la logica (si potrebbe partire da Aristotele) e poi trattate con rigore matematico da Boole, giungendo alla loro formalizzazione.

L'algebra di Boole studia le operazioni su insiemi composti da due soli elementi, che indicheremo per brevità coi simboli "0" e "1".

> ***Nota***
>
> *I simboli "0" e "1" usati tradizionalmente nell'algebra di Boole, non hanno nulla a che vedere né con le cifre né con i numeri corrispondenti, potrebbero essere sostituiti da due altri simboli qualsiasi: infatti in altri contesti si utilizza al loro posto "T" (True) e "F" (False) oppure "H" (High) e "L" (Low).*

Pertanto una variabile "booleana" (rappresentata con una lettera maiuscola, come I, U, etc.) può assume solo due valori, 0 oppure 1. L'algebra booleana è anche chiamata *algebra binaria* (binaria = applicabile ad un insieme di due elementi).

L'importanza dell'algebra di Boole sta nel fatto che essa costituisce il modello matematico per svariate situazioni, per esempio la logica. I circuiti elettronici digitali (detti anche circuiti o porte logiche) eseguono fisicamente le operazioni booleane e costituiscono i mattoni con cui si sono costruite macchine sempre più complesse, fino agli attuali computer digitali.

Cominciamo definendo una *Macchina Astratta Booleana*, rappresentata come una *scatola nera* (ossia una scatola di cui non conosciamo il contenuto, ma di cui possiamo solo osservare le interazioni con l'ambiente in cui viene posta) composta da N ingressi ed M uscite di tipo boolean come rappresentato in figura:

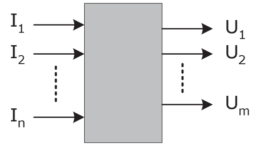


Fig. 1.1 - Macchina Booleana combinatoria

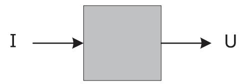
In particolare, definiamo Macchina Booleana *combinatoria* una scatola nera per la quale i valori di una qualsiasi uscita (U_j) sono funzione solo degli ingressi, ossia in simboli:

$$U_j = f_j(I_1, I_2, \ldots I_n)$$

Studiamo il problema cominciando dal caso più semplice: n=1. Si tratta di funzioni di una sola variabile:

$$U = f(I)$$

Per definire una funzione, basta definirne il risultato per ogni possibile combinazione dei valori delle variabili (*dominio di definizione*). Nel caso dell'algebra di Boole questo è un processo semplice, perché sia le variabili che il risultato possono assumere solo due valori, 0 oppure 1. Si possono quindi esaminare tutte le funzioni di una sola variabile semplicemente elencandole: occorre definire 2 risultati (uno per ogni valore della variabile), e poiché ogni risultato può assumere 2 valori ("0" oppure "1") in tutto si hanno $2^2 = 4$ funzioni di una sola variabile (dette anche *funzioni unarie*).



fa Variabile I	*fa* Risultato U	*fb* Variabile I	*fb* Risultato U	*fc* Variabile I	*fc* Risultato U	*fd* Variabile I	*fd* Risultato U
0	0	0	0	0	1	0	1
1	0	1	1	1	0	1	1

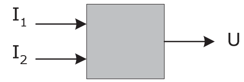
Fig. 1.2 - Funzioni Booleane unarie

La prima e l'ultima (f_a e f_d) sono *costanti* (infatti il valore in uscita é costante, indipendentemente dal valore in ingresso). La seconda funzione (f_b) é l'*identità* (infatti il valore in uscita é identico al valore in ingresso), quindi non particolarmente utile per la realizzazione di sistemi di calcolo. La terza funzione (f_c) é l'unica funzione booleana di una variabile "interessante" dal nostro punto di vista: la chiameremo *funzione negazione (***NOT***)*, visto che il valore in uscita é sempre il valore opposto rispetto a quello in ingresso.

Passiamo al caso di funzioni due variabili cioè del tipo:

$$U = f(I_1, I_2)$$

Si possono esaminare tutte le funzioni di due variabili elencandole: occorre definire 4 risultati (uno per ogni combinazione delle variabili), e poiché ogni risultato può assumere 2 valori ("0" oppure "1") in tutto si hanno $2^4 = 16$ funzioni di due variabili (dette anche *funzioni binarie*). Le raggruppiamo in un'unica tabella per comodità:



variabili		funzioni															
I_1	I_2	f_e	f_f	f_g	f_h	f_i	f_l	f_m	f_n	f_o	f_p	f_q	f_r	f_s	f_t	f_u	f_v
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
			NOR					XOR	NAND	AND	NXOR					OR	

Fig. 1.3 - Funzioni Booleane binarie

Alcune di queste 16 funzioni non sono interessanti: la prima (f_e) e l'ultima (f_v) sono costanti, ossia specificano una uscita indipendente dai valori di ingresso; due (f_q e f_s) sono funzioni identità su una sola delle variabili di ingresso; altre due (f_h e f_l) sono funzioni negazione su una sola delle variabili in ingresso.

Le rimanenti 10 funzioni sono invece non banali. Nel nostro contesto solo alcune acquistano particolare importanza, ricevendo un nome: f_o viene chiamata *funzione and* o *congiunzione logica* (AND), f_m viene chiamata *funzione or esclusivo* (XOR), f_u viene chiamata *funzione or inclusivo* o *disgiunzione logica* (OR). Sono anche interessanti le funzioni ottenute negando il risultato delle precedenti: f_f (NOR), f_n (NAND) e f_p (NXOR).

Nota

In altri contesti possono essere importanti altre funzioni, e.g. in logica matematica la funzione f_r ha il nome di condizionale (IF), …

Non è necessario andare oltre: qualsiasi funzione logica, indipendentemente dall'*arità* (numero di operandi), è scomponibile in funzioni di due variabili, cioè nelle funzioni già viste.

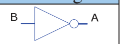
1.3.1 LE ESPRESSIONI BOOLEANE

La composizione di operazioni binarie in espressioni booleane complesse si può ottenere utilizzando differenti notazioni il cui utilizzo è funzione del contesto utilizzato. Nel capitolo verranno descritti i seguenti formalismi:

- la *notazione algebrica*, molto simile a quella dell'algebra tradizionale;
- la *notazione logica,* diffusa, a meno della sintassi dei singoli operatori, nella maggior parte dei linguaggi di programmazione;
- le sintassi dei *linguaggi di programmazione: JAVA e C;*
- la sintassi del linguaggio di presentazione e trasformazione di dati *XML (XSTL, eXtensible Stylesheet Language Transformations*);
- la rappresentazione grafica dei corrispondenti componenti elettronici digitali.

Associamo anche alle varie funzioni delle proprietà (assiomi o teoremi dell'algebra di Boole, a seconda dell'impostazione usata) che utilizzeremo praticamente per semplificare le espressioni.

NEGAZIONE: NOT

n. algebrica	n. logica	JAVA	XSLT	C	circuito digitale
$A = \overline{B}$	$A = \neg B$	A = !B	not(B)	A = !B	

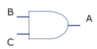
Tab. 1.1 – Operazione NOT

Definizione algoritmica: *«il risultato di un NOT è 1 se l'operando vale 0 altrimenti il risultato è 0».*

PROPRIETÀ:

p_1 eliminazione della doppia negazione $A = \overline{\overline{A}}$

CONGIUNZIONE: AND

n. algebrica	n. logica	JAVA	XSLT	C	circuito digitale
$A = B \cdot C$	$A = B \wedge C$	A= B & C	B and C	A= B && C	

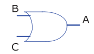
Tab. 1.2 – Operazione AND

Definizione algoritmica (estesa a n operandi): *«il risultato di un AND è 1 solo se **tutti** gli operandi valgono 1».*

PROPRIETÀ:

p_2	idempotenza	A · A = A
p_3	commutativa	A · B = B · A
p_4	associativa	A · (B · C) = (A · B) ·C
p_5	elemento neutro	1 · A = A
p_6	elemento assorbente	0 · A = 0

DISGIUNZIONE: OR

n. algebrica	n. logica	JAVA	XSLT	C	circuito digitale
$A = B + C$	$A = B \vee C$	A= B | C	B or C	A= B || C	

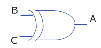
Tab. 1.3 – Operazione OR

Definizione algoritmica (estesa a n operandi): *«il risultato di un OR è 1 se **almeno un** operando vale 1».*

PROPRIETÀ:

p_7	idempotenza	A + A = A
p_8	commutativa	A + B = B + A
p_9	associativa	A + (B + C) = (A + B) + C
p_{10}	elemento neutro	0 + A = A
p_{11}	elemento assorbente	1 + A = 1

OR ESCLUSIVO: XOR

n. algebrica	n. logica	java	XSLT	C	circuito digitale
$A = B \oplus C$	$A = B \oplus C$	A= B ^ C	*N.A.*	*N.A.*	

Tab. 1.4 – Operazione XOR

Definizione algoritmica (estesa a n operandi): *«il risultato di un XOR è 1 se **un numero dispari** di operandi valgono 1».*

PROPRIETÀ:

p_{12} Conversione $A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$

ALTRE PROPRIETÀ:

p_{13} distributività di AND rispetto a OR
A·(B + C)= (A·B) + (A·C)

p_{14} distributività di OR rispetto a AND
A+(B·C)= (A+B)· (A+C)

p_{15} AND di complementi $A \cdot \overline{A} = 0$

p_{16} OR di complementi $A + \overline{A} = 1$

p_{17} Leggi di De Morgan $\overline{A\cdot B} = \overline{A} + \overline{B}$

$\overline{A+B} = \overline{A}\cdot\overline{B}$

> ***Nota***
>
> *Attenzione AND e OR nulla hanno a che vedere con il prodotto o la somma aritmetica, nonostante i simboli simili. La scelta (infelice) di tali simboli per le operazioni logiche è storica e dovuta ad alcune similitudini formali*

EQUIVALENZE

La legge di De Morgan dimostra che è possibile trasformare una operazione di congiunzione logica nella corrispettiva disgiunzione logica, e quindi che si può esprimere l'operatore AND in funzione di OR e, viceversa, OR in termini di AND.

In relazione alle proprietà di idempotenza degli operatori binari AND ed OR (p_2 e p_7) è anche possibile esprimere l'operazione unaria di negazione con gli operatori NAND e NOR:

$$\overline{A\cdot A} = \overline{A}$$

$$\overline{A+A} = \overline{A}$$

D'altra parte sia la congiunzione logica, che la disgiunzione logica, si possono esprimere in termini di NAND (o NOR):

NAND

$\overline{\overline{A\cdot B}\cdot\overline{A\cdot B}} = \overline{\overline{A\cdot B}} = A\cdot B$ $\qquad$ $\overline{\overline{A\cdot A}\cdot\overline{B\cdot B}} = \overline{\overline{A}\cdot\overline{B}} = A+B$

NOR

$\overline{\overline{A+B}+\overline{A+B}} = \overline{\overline{A+B}} = A+B$ $\qquad$ $\overline{\overline{A+A}+\overline{B+B}} = \overline{\overline{A}+\overline{B}} = A\cdot B$

Alla luce di queste considerazioni è possibile concludere che l'operatore NAND oppure l'operatore NOR consentono di implementare tutte le funzioni binarie e, pertanto, sono detti *operatori universali*. Disponendo quindi di un solo circuito (NAND oppure NOR) è possibile realizzare qualsiasi macchina combinatoria.

Nelle notazioni qui presentate sono sempre ammesse parentesi per esplicitare l'ordine delle operazioni, e, come per gli analoghi operatori dell'aritmetica, l'operatore NOT ha la maggiore priorità rispetto a tutti gli altri mentre seguono nell'ordine l'operatore AND e l'operatore OR.

Le regole di precedenza riducono la necessità di parentesi.

Quindi l'espressione: $X = \overline{((A\cdot B)+C)}$

Si può anche scrivere: $X = \overline{A\cdot B + C}$

> ***Nota***
>
> *Esistono altri sistemi di notazione che non richiedono l'uso di parentesi, per esempio la "notazione polacca inversa" (RPN, Reverse Polish Notation) quella di J. Lukasiewicz e la notazione del Dienes. Alcuni di questi sistemi di notazione sono importanti ed ampiamente utilizzati in programmazione (vedi oltre §9.2.7).*

Esempio

L'espressione precedente si può scrivere (Lukasiewicz): X = **N A K** A B C

Il significato dei simboli utilizzati da Lukasiewicz risulta immediatamente se scriviamo nella forma:

NOT (OR (AND (A , B),C))

È una notazione prefissa in cui il simbolo dell'operazione precede gli operandi (uno o due a seconda dell'arità). Prima si calcola A AND B (**K** A B) poi il risultato è in OR con C (**A K** A B C) infine si nega il tutto (**N A K** A B C).

Uno strumento molto utilizzato per definire nuove funzioni booleane, per effettuare dimostrazioni o verifiche, è la tabella o *tavola di verità*. Questa tabella rappresenta, alla sinistra di una doppia riga verticale, con una colonna per variabile, tutti i possibili valori degli operandi ed a destra i valori assunti dalla funzione od espressione, utilizzando una colonna o più colonne.

Esempio

V*erificare la proprietà di conversione dell'XOR (p_{12}):* $A\oplus B = \overline{A}\cdot B + A\cdot\overline{B}$

a) Si inizia con una tabella contenente tutte le combinazioni (2^n) delle n variabili, una per riga:

variabili		1	2	3	4	5	6
A	B						
0	0						
0	1						
1	0						
1	1						

Tab. 1.5 – Tavola di verità: passo 1

b) Nella colonna (1) si inseriscono i risultati relativi alla parte sinistra dell'equazione, calcolati riga per riga applicando la definizione (vedi f_m):

variabili		1	2	3	4	5	6
A	B	$A\oplus B$					
0	0	0					
0	1	1					
1	0	1					
1	1	0					

Tab. 1.6 – Tavola di verità: passo 2

c) *Poi si calcola la parte destra dell'espressione usando delle colonne ausiliarie (2..5) ed applicando un'operazione alla volta, riga per riga. La colonna 2 è ottenuta negando B, la colonna 3 è il risultato dell'AND tra A e la colonna 2 ,* *la 4 si ottiene negando A, la 5 eseguendo un AND tra 4 e B, la colonna 6 è il risultato di un OR tra le colonne 3 e 5.*

L'ordine di precedenza tra gli operatori è quello usuale: prima le espressioni tra parentesi, l'AND ha precedenza su OR …

Variabili		1	2	3	4	5	6
A	B	$A\oplus B$	$\overline{B}$	$A\cdot\overline{B}$	$\overline{A}$	$\overline{A}\cdot B$	$A\cdot\overline{B}+\overline{A}\cdot B$
0	0	0	1	0	1	0	0
0	1	1	0	0	1	1	1
1	0	1	1	1	0	0	1
1	1	0	0	0	0	0	0

Tab. 1.7 – Tavola di verità: verifica

d) Conclusione: poiché in tutte le righe il valore della colonna 1 è uguale a quello della colonna 6, si è verificata la proprietà.

Notare che questo stile di dimostrazione, che verifica tutti i casi possibili, è applicabile e valido per l'algebra di Boole solo perché l'insieme di definizione è finito. Nel caso dell'algebra classica, essendo i numeri infiniti, si possono effettuare dimostrazioni solo tramite deduzione o induzione dai teoremi.

1.3.2 LA TRASFORMAZIONE DI ESPRESSIONI BOOLEANE

Una espressione logica può essere scritta in infiniti modi diversi, tra loro equivalenti. Sono stati sviluppati diversi metodi per semplificare le espressioni booleane (*metodo di Quine-McCluskey*, *mappe di Karnaugh*, …) ma quello più flessibile rimane il metodo algebrico.

Questo consiste nel passare da una espressione booleana ad un'altra equivalente applicando le proprietà elencate precedentemente, in un modo analogo al procedimento utilizzato per risolvere un'equazione algebrica tradizionale.

Passare dalla equazione al circuito elettronico o viceversa è banale. Interessante è invece il passaggio dalla tavola di verità ad una equazione booleana equivalente.

E' infatti relativamente semplice affrontare un problema di logica definendo la sua tavola di verità.

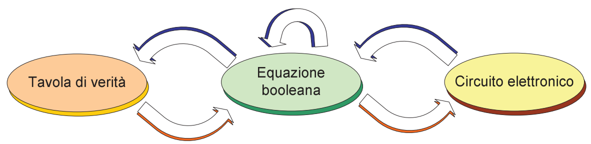


Fig. 1.4 – Schema riassuntivo per la trasformazione delle espressioni logiche

Esempio

Consideriamo un semplice sistema di allarme. Un istituto bancario ha due porte blindate per l'accesso ai locali della clientela; ipotizziamo di utilizzare due sensori elettronici che rilevano l'apertura delle due porte:

- *sensore A = 1 se la prima porta è aperta;*
- *sensore B = 1 se la seconda porta è aperta.*

Inoltre potremmo assicurare il passaggio di personale autorizzato attraverso le due porte solo su volontà di un responsabile dell'istituto mediante l'attivazione di un interruttore o di una scheda magnetica (variabile logica C):

- *C = 1 se il responsabile ha disattivato il sistema di allarme;*
- *il risultato è rappresentato dalla variabile X, scegliamo che in corrispondenza del valore X = 1 l'allarme deve suonare.*

Definiamo il comportamento desiderato utilizzando una tavola di verità: a sinistra tutte le combinazioni delle variabili (3 variabili, $2^3 = 8$ righe):

C	A	B	Descrizione	X
0	0	0	Entrambe le porte sono chiuse, l'allarme è inserito	0
0	0	1	La porta B è aperta, l'allarme è inserito	0
0	1	0	La porta A è aperta, l'allarme è inserito	0
0	1	1	Le porte A e B sono aperte, l'allarme è inserito	1
1	0	0	Entrambe le porte sono chiuse, l'allarme è spento	0
1	0	1	La porta B è aperta, l'allarme è spento	0
1	1	0	La porta A è aperta, l'allarme è spento	0
1	1	1	Le porte A e B sono aperte, l'allarme è spento	0

Tab. 1.8 – Tavola di verità: definizione di funzione

L'algoritmo per ricavare dalla tavola l'espressione booleana risulta il seguente:

«si considerano i valori 1 nella colonna che si vuole sintetizzare (X), e si scrive un gruppo di variabili per ogni 1. Ciascun gruppo (chiamato anche MINTERM) è formato da tutti gli operandi (A, B e C) legati dall'operatore AND, in cui ciascun operando risulta negato (operatore NOT) se nella corrispondente riga assume il valore 0. I gruppi individuati sono legati da OR. L'espressione che si ottiene corrisponde alla tavola di verità di partenza ed è anche chiamata *forma normale* (o *canonica*) *congiuntiva*, ed è unica».

Nell'esempio c'è un solo gruppo, quindi: $X=\overline{C}\cdot A\cdot B$

Proprio in virtù del teorema di De Morgan esiste un algoritmo duale per la sintesi di una espressione logica da tavola di verità:

«si considerano i valori 0 nella colonna che si vuole sintetizzare (X), e si scrive un gruppo di variabili per ogni 0. Ciascun gruppo è formato da tutti gli operandi (A, B e C) legati dall'operatore OR, in cui ciascun operando risulta negato (operatore NOT) se nella corrispondente riga assume il valore 1. I gruppi individuati sono legati da AND. L'espressione che si ottiene corrisponde alla tavola di verità di partenza ed è anche chiamata *forma normale* (o *canonica*) *disgiuntiva*, ed è unica».

Nell'esempio i gruppi di AND sono sette, quindi:

$$X = (C+A+B)\cdot(C+A+\overline{B})\cdot(C+\overline{A}+B)\cdot(\overline{C}+A+B)\cdot$$
$$\cdot(\overline{C}+A+B)\cdot(\overline{C}+A+\overline{B})\cdot(\overline{C}+\overline{A}+B)\cdot(\overline{C}+\overline{A}+\overline{B})$$

E' facile concludere che, per ottenere una espressione con il minor numero di termini, è necessario considerare il minor numero di simboli (zero o uno).

Nota

Esistono programmi sui PC che eseguono automaticamente le operazioni di trasformazione tra tavola di verità, equazione e circuito.

1.3.3 CONCLUSIONE

L'algebra di Boole permette di definire funzioni qualsiasi di n variabili ed i componenti digitali elettronici permettono di realizzare circuiti con comportamento definito da equazioni booleane.

Esercizio

Sviluppare un sistema per la rilevazione delle radiazioni gamma che invii un messaggio di allarme quando il valore della radiazione gamma eccede il valore γ_{min}*, in presenza di pioggia; a causa dell'innalzamento del valore della radiazione, il valore di tolleranza del sistema è incrementato al valore* γ_{max}*. Implementare la funzione logica A(t) che descrive il funzionamento del sistema al variare degli input.*

Il sistema è caratterizzato dalle seguenti sorgenti in ingresso: la misura della radiazione all'istante t, γ(t), e l'evento pioggia al variare di t, P(t). In uscita il sistema fornisce il valore dello stato dell'allarme nel tempo, A(t).

A questo punto occorre attribuire un significato ai valori 0 e 1, supponiamo di assumere che: P(t)=1 corrisponda all'evento pioggia, mentre A(t)=1 all'evento allarme.

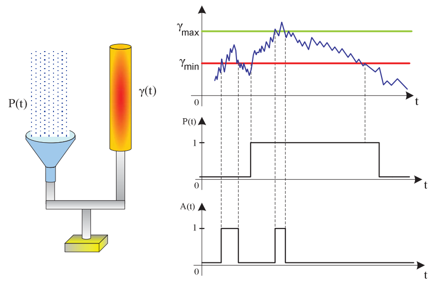


Fig. 1.5 - Sistema per la rilevazione delle radiazioni gamma

Attribuiamo ora alle variabili logiche le seguenti condizioni:

$X = \gamma(t) > \gamma_{min}$ $\qquad$ $Y = \gamma(t) > \gamma_{max}$

X	Y	P	A
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Tab. 1.9 – Tavola di verità per rilevatore radiazioni Gamma

A questo punto l'espressione cercata diventa:

$X\cdot\overline{Y}\cdot\overline{P}+X\cdot Y\cdot\overline{P}+X\cdot Y\cdot P=$ (*forma canonica dalla tavola di verità*)

$X\cdot\overline{Y}\cdot\overline{P}+X\cdot Y\cdot\overline{P}+X\cdot Y\cdot\overline{P}+X\cdot Y\cdot P=$ (*applicazione della proprietà* p_7)

$X\cdot\overline{P}(\overline{Y}+Y)+X\cdot Y(\overline{P}+P)=$ (*applicazione della proprietà* p_{13})

$X\cdot\overline{P}\cdot 1+X\cdot Y\cdot 1=$ (*applicazione della proprietà* p_{16})

$X\cdot\overline{P}+X\cdot Y=$ (*applicazione della proprietà* p_5)

$X\cdot(\overline{P}+Y)$ (*applicazione della proprietà* p_{13})

1.4 GLI AUTOMI

Gli automi sono delle macchine sequenziali in cui l'uscita non è solo funzione degli ingressi attuali, come in una macchina combinatoria, ma anche della storia passata. Per esempio, una sveglia suonerà al mattino all'ora in cui è stata impostata la sera. Il concetto di “storia passata” è corretto ma di scomodo uso: si preferisce usare il concetto di “stato”, definito come l’informazione necessaria e sufficiente per riassumere la storia passata. Una macchina sequenziale (a stati finiti) può essere modellata in due modi equivalenti.

Definiamo: S l’insieme di stati, I l’insieme dei possibili ingressi, U l’insieme delle uscite.

La *Macchina di Mealy* è definita da una quintupla (*S, I, U, t(), w()*), dove *t()* è una funzione dello stato attuale e dell’ingresso che permette di calcolare lo stato successivo, e *w()* è invece la funzione che da stato ed ingresso permette di ricavare l’uscita. Analogo è il modello chiamato *Macchina di Moore*: la quintupla (*S, I, U, t(), u()*) è simile alla precedente, ma ora la funzione *u()* calcola l’uscita in funzione del solo stato. Questi modelli sono equivalenti e si possono applicare a molti sistemi, tra cui anche i circuiti elettronici.

Esempio

Un latch SR è forse il più semplice circuito digitale rappresentabile come automa sequenziale. Gli ingressi siano S (Set) e R (Reset). L’uscita è Q.

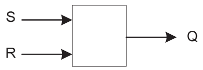


Fig. 1.6 - Latch SR

La caratterizzazione del latch SR tramite il modello di Moore è la seguente:

Input: sono ammesse le seguenti coppie (S,R): (0,0) (0,1) (1,0)

Stati: il Latch ha due stati, γ_0 e γ_1, per la loro codifica è sufficiente un bit.

Uscita: Q assume due valori, 0 e 1.

La funzione *u()* che lega l’uscita allo stato è la funzione identità, rappresentata dalla tavola:

stato	Uscita Q
γ_0	0
γ_1	1

Tab. 1.10 – Tavola di verità per latch SR

La funzione *t()* è invece definita tramite la tavola che lega lo stato successivo (*γ’*) allo stato attuale (γ) ed agli ingressi (S e R).

stato attuale	Input (SR)		
	00	01	10
γ_0	γ_0	γ_0	γ_1
γ_1	γ_1	γ_0	γ_1

Tab. 1.11 – Tavola degli stati per latch SR

La realizzazione di un circuito elettronico sequenziale si può effettuare utilizzando due circuiti combinatori (uno per la funzione *t()* e uno per *u()*) ed un ritardo.

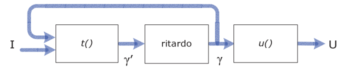


Fig. 1.7 – Schema di un circuito elettronico sequenziale

Per realizzare la funzione t() scriviamo una tavola di verità, come già fatto per i circuiti combinatori, ricavandola dalla tabella precedente:

S	R	γ	γ’
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

Tab. 1.12 – Tavola di verità per la funzione *t()*

Da questa si ha l’equazione:

$$Q' = S + Q \cdot \overline{R}$$

Questo è il circuito corrispondente (il ritardo dei circuiti logici è sufficiente, non è necessario nessun ritardo aggiuntivo):

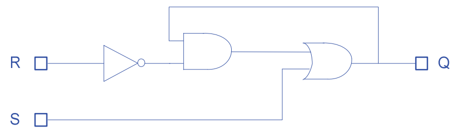


Fig. 1.8 – Rappresentazione del circuito elettronico sequenziale

Per gli automi a stati finiti, oltre alla definizione mediante tabelle, è possibile un’altra definizione mediante un grafo. Gli stati e le uscite sono rappresentati da rettangoli arrotondati, mentre le transizioni sono rappresentate da frecce, con l’indicazione degli input che causano la transizione.

Questa è rappresentazione del latch SR, equivalente alla rappresentazione tabellare precedente.

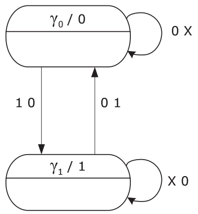


Fig. 1.9 – Rappresentazione mediante grafo del lacth SR

Sulle frecce sono indicati i due bit di input, S ed R, nell'ordine, ed 'X' indica sia 0 che 1. Le etichette dei due stati, γ_0 e γ_1, indicano anche le uscite.

1.5 I SISTEMI DI NUMERAZIONE

Il metodo che usiamo normalmente per rappresentare i numeri interi è definito *sistema posizionale in base 10*. Come la definizione suggerisce possono esistere sistemi non posizionali e sistemi posizionali in base diversa da 10.

Per esempio il sistema utilizzato dagli antichi romani è un sistema non posizionale: l'anno 2002 sarebbe stato scritto dai romani come

MMII

dove ogni M vale 1000 e le I valgono uno.

Nota

Nel sistema romano sono presenti gli echi del contare con le dita:

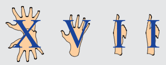
17 = XVII = 

Fig. 1.10 – Numerazione romana

Il sistema attuale si deve agli Arabi ed è stato reso possibile dall'introduzione dello zero, per indicare un posto libero. Il vantaggio che presenta è la semplicità degli algoritmi di calcolo per le quattro operazioni, che si effettuano cifra per cifra.

Come è strutturato un sistema posizionale? Sia B la base: si usano B simboli (cifre) differenti, da 0 a B-1 per rappresentare quantità minori della base. Il *peso* (il contributo al totale) di ogni cifra dipende dalla posizione della cifra all'interno del numero: normalmente (in base 10) diciamo cifra delle unità, delle decine, delle centinaia, etc.

In generale il peso è pari ad una potenza della base, crescente da destra verso sinistra, cominciando dall'esponente zero.

Esempio

Rappresentare 2002 in base 10.

cifre	**2**	**0**	**0**	**2**
pesi	10^3	10^2	10^1	10^0
cifre x pesi	2000	0	0	2

Questo meccanismo è del tutto generale e può essere usato con qualunque base.

Se la base è maggiore di 10, dopo le cifre usuali 0..9, sono usate come cifre le lettere dell'alfabeto A(10), B(11), C(12), D(13), E(14), F(15),…

Nota

In alcuni settori merceologici si usa normalmente la base 12. In questi casi si parla di dozzine e di grosse (dozzine di dozzine): una grossa di calze è pari a 144 paia ($1 \times 12^2 = 144$) di calze.

Nei successivi paragrafi verranno affrontati i seguenti problemi:

- come passare da base 10 ad una base qualunque e viceversa;
- come eseguire operazioni in una base qualsiasi.

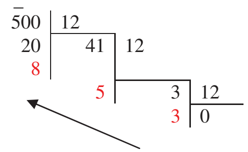
1.5.1 PASSAGGIO DA BASE 10 A BASE QUALSIASI

Per eseguire il passaggio da base 10 ad una qualsiasi base si utilizza il *metodo delle divisioni* successive che verrà descritto attraverso un esempio.

Esempio

Trasformare 500_{10} in base 12.

a) Dividere il numero per la nuova base, e continuare, dividendo ancora il risultato, fino ad ottenere zero come risultato.



b) I resti, presi nell'ordine indicato dalla freccia, sono le cifre cercate, pertanto si può scrivere (la base si deve sempre indicare nei casi in cui è possibile fare confusione):

$$500_{10} = 358_{12}$$

cifre	3	5	8
pesi	12^2	12^1	12^0
cifre x pesi	432	60	8

Ovvero: 3 grosse, 5 dozzine e 8 unità.

Note

- in ogni base, la base stessa si scrive 10_b (in base 12, 10 si scrive 10_{12});
- in basi diverse da 10 i numeri si leggono come cifre separate: 500_{10} si legge "cinquecento" ma 358_{12} si legge "tre-cinque-otto in base dodici".

Esempio
Trasformare 500_{10} in base 16.

```
500 | 16
 20 | 31 | 16
  4 |    |  1 | 16
    | 15 |    |  0
    |    |  1
```

$500_{10} = 1F4_{16}$

1.5.2 PASSAGGIO DA BASE QUALSIASI A BASE 10

Per eseguire il passaggio da una base qualsiasi a base 10, supponendo il numero in base b, X_b, costituito dalle cifre:

$$X_b = \quad | \; c_k \quad c_{k-1} \quad c_{k-2} \quad \ldots \quad c_2 \quad c_1 \quad c_0$$

si utilizza la formula:

$$N_{10} = \sum_{0}^{k} {}_j c_j \; b^j$$

che esprime il numero nella base b come somma di prodotti, in cui ciascun termine dipende dalla cifra c_j (tradotta in base 10) e dalla base b elevata al valore j.

Esempio
Trasformare 358_{12} in base 10.

Applicando la definizione di sistema posizionale, calcolare i pesi e il contributo di ogni cifra in base 10 con lo schema di calcolo seguente:

$$
\begin{array}{llcr}
8 \times 12^0 & = 8 \times 1 & = & 8 \; + \\
5 \times 12^1 & = 5 \times 12 & = & 60 \; + \\
3 \times 12^2 & = 3 \times 144 & = & \underline{432 \; =} \\
 & & & 500
\end{array}
$$

1.5.3 IL SISTEMA DI NUMERAZIONE BINARIO (BASE 2)

La base 2 ha una particolare importanza in informatica, e pur valendo tutte le considerazioni generali precedenti, si possono applicare alla base 2 alcune utili semplificazioni.

Iniziamo fornendo una tavola delle potenze di 2: col tempo diverranno familiari ma all'inizio è opportuno averle sottomano:

2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1.024
2^{11}	2.048
2^{12}	4.096
2^{13}	8.192
2^{14}	16.384
2^{15}	32.768
2^{16}	65.536
2^{17}	131.072
2^{18}	262.144
2^{19}	524.288
2^{20}	1.048.576

Tab. 1.13 – Potenze di 2

Nota

Come già visto, 2^n è il numero di righe in una tavola della verità di n variabili, cioè il numero delle disposizioni con ripetizione di 2 simboli in classe n.

1.5.4 PASSAGGIO DA BASE 10 A BASE 2 E VICEVERSA

Esempio
Trasformare 50_{10} in base 2.

Si usa una versione semplificata del metodo generale:

a) A sinistra di una riga verticale scrivere il numero da convertire, a destra scrivere il resto: 0 se il numero è pari, 1 se è dispari. Sotto scrivere il risultato della divisione (fatta a mente) del numero per due.

50	0
25	

b) Ripetere fino ad ottenere 0 come risultato a sinistra.

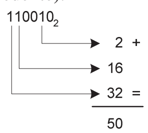
50	0
25	1
12	0
6	0
3	1
1	1
0	

Prendere i resti nell'ordine indicato dalla freccia.

$$50_{10} = 110010_2$$

Esempio
Trasformare 110010_2 in base 10.

Si usa una versione semplificata del metodo generale, sommando le sole potenze di 2 corrispondenti agli "1" presenti nel numero in base 2 (le potenze di due si possono calcolare mentalmente partendo da 1 e raddoppiando ogni volta, oppure si può usare la tabella precedente):



1.5.5 IL SISTEMA DI NUMERAZIONE ESADECIMALE (BASE 16)

La base 16 è molto utilizzata soprattutto come "stenografia" per la base 2. Un tempo si usava anche la base 8 con i medesimi scopi, ma è caduta in disuso con la diffusione delle stampanti alfanumeriche. Il linguaggio Java la utilizza ancora oggi per la codifica dei caratteri fino al valore 255 (vedi oltre §5.5).

Un numero binario di 4 cifre è rappresentabile più semplicemente con una singola cifra esadecimale. Vediamo ciò in una tabella:

base 10	***base 16***	***base 2*** pesi: 8421
0	**0**	0000
1	**1**	0001
2	**2**	0010
3	**3**	0011
4	**4**	0100
5	**5**	0101
6	**6**	0110
7	**7**	0111
8	**8**	1000
9	**9**	1001
10	**A**	1010
11	**B**	1011
12	**C**	1100
13	**D**	1101
14	**E**	1110
15	**F**	1111

Tab. 1.14 – Tabella di conversione rapida decimale - esadecimale – binario

Esempio
Trasformare $B800_{16}$ in base 2.

Questa conversione (e quella inversa) può essere fatta cifra per cifra usando la tabella precedente. Ad ogni quattro cifre binarie corrisponde una cifra esadecimale.

B	**8**	**0**	**0**
1011	*1000*	*0000*	*0000*

$$B800_{16} = 1011100000000000_2$$

Ovviamente il numero esadecimale è molto più comodo da maneggiare rispetto al suo corrispondente binario poiché a parità di informazione il numero di cifre è ridotto di un fattore quattro (la proporzione che esiste tra gli esponenti della base).

Nota
Questa tecnica di conversione (cifra per cifra) vale solo per basi che siano l'una potenza dell'altra: in generale per passare da una base qualsiasi ad un'altra si deve sempre passare per la base 10.

1.5.6 LE QUATTRO OPERAZIONI IN UNA BASE QUALSIASI

Completiamo l'analisi dei sistemi di numerazione analizzando come effettuare le quattro operazioni aritmetiche con numeri in base diversa da 10.

Una buona notizia: si fanno esattamente come in base 10, usando gli stessi algoritmi, cifra per cifra, con riporti, etc.

Con una sola avvertenza: nel calcolare un risultato noi ragioniamo e calcoliamo in base 10, ma per scrivere una cifra o un riporto, si deve convertire e scrivere nella base in uso. Idem per le "cifre prese in prestito".

Esempio

Calcolare $1F4_{16} + 1F4_{16}$

$$\begin{array}{r} 1\phantom{F4_{16}} \\ 1F4_{16}\ + \\ 1F4_{16}\ = \\ \hline 3E8_{16} \end{array}$$

a) si analizza la prima colonna a destra e si esegue l'addizione delle cifre (4+4 = 8);

b) si passa alla seconda colonna e si procede come nel passaggio precedente:
$$F_{16} + F_{16} = 15_{10} + 15_{10} = 30_{10} = 1E_{16}$$
il risultato produce due cifre: la meno significativa (E_{16}) si scrive mentre l'altra (1_{16}) costituisce il riporto per la successiva colonna;

c) si passa alla terza colonna e si procede come nel passaggio precedente:
$$1_{16} + 1_{16} + 1_{16} = 3_{16}$$
il risultato produce un valore che viene scritto in colonna (la cifra 3_{16}).

Esempio

Calcolare $1011_2 - 101_2 \quad (11_{10} - 5_{10} = 6_{10})$

$$\begin{array}{r} 1 \\ 1011_2\ - \\ 101_2\ = \\ \hline 110_2 \end{array}$$

d) si analizza la prima colonna a destra e si esegue la sottrazione delle cifre:
$$1 - 1 = 0;$$

e) si passa alla seconda colonna e si procede come nel passaggio precedente:
$$1 - 0 = 1;$$

f) si passa alla terza colonna e si procede come nel passaggio precedente, in questo caso l'operazione è impossibile quindi occorre prendere un 1 dalla posizione successiva (vale la base, cioè 2):
$$2_{10} + 0_{10} - 1_{10} = 1_{10}$$
il risultato produce il valore 1_{10} ($=1_2$) che viene scritto in colonna;

g) si passa alla quarta colonna e si procede come nel passaggio precedente:
$$1 - 1 \text{ (preso in prestito)} = 0$$
il risultato è il valore 0, non si scrive nulla.

1.6 LE OPERAZIONI DEL COMPUTER

Per utilizzare i circuiti elettronici digitali per effettuare calcoli aritmetici si usa la base 2: è un sistema di numerazione che usa solo due cifre (0 e 1) e quindi si possono utilizzare i due valori dell'algebra di Boole per rappresentarle. Ogni cifra di un numero binario è quindi rappresentabile con una variabile booleana, e la cifra sarà 0 o 1 a seconda del valore della variabile.

Esempio

Progettare un circuito per eseguire la somma di due numeri binari (full adder)

Ridisegnamo lo schema precedente della somma in base 2 per definire i nomi di alcune variabili:

$$\begin{array}{ccc} C_1 & C_0 & \\ \dots & A_0 & \dots \\ \dots & B_0 & \dots \\ \hline \dots & S0 & \dots \end{array}$$

Per sommare due cifre binarie (A_0 e B_0) occorre anche sommare il possibile riporto (C_0) della somma delle cifre a destra. Si deve calcolare la cifra del risultato (S0) e la cifra dell'eventuale riporto a sinistra (C_1).

Possiamo definire i risultati desiderati, come definiti dalle leggi dell'addizione binaria, tramite una tavola di verità:

variabili			risultato	riporto
C_0	B_0	A_0	S_0	C_1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tab. 1.15 – Tavola di verità per "full adder"

Aumentando il numero di cifre in ingresso ed il tipo di operazioni eseguibili si giunge ad un componente alquanto complesso, chiamato ALU (*Arithmetic Logic Unit*), disponibile sia come componente separato che inserito all'interno dei microprocessori, in grado di eseguire tutte le operazioni matematiche elementari su più cifre binarie.

1.6.1 BIT, BYTE E SIMILI

Per gestire numeri interi in base 2 occorrono dunque più variabili logiche booleane, una per ciascuna cifra.
Più variabili si usano, più grande può essere il numero rappresentato: esattamente, con N variabili booleane si possono rappresentare fino a 2^N valori diversi. Per ragioni tecniche e storiche, si sono privilegiati gruppi di cifre potenza di 2, tanto da assumere nomi particolari:

N	2^N	nome	Note
1	2	bit	Binary digIT, o cifra binaria
4	16	nibble	può essere rappresentato con una singola cifra esadecimale
8	256	Byte	raggruppamento fondamentale, usato come unità di misura
16	65.536	WORD	corrisponde a 2 Byte
32	4.294.967.296	DOUBLE WORD	corrisponde a 4 Byte

Tab. 1.16 – Definizioni

Per rappresentare i numeri interi relativi (vale a dire con segno) in base 2 viene comunemente usata una tecnica chiamata "complemento a 2". Questo è un caso particolare del "complemento alla base", metodo che può essere utilizzato con tutte le basi, compreso la base 10 ("complemento a 10"). Il vantaggio che offre questo metodo consiste nel poter eseguire con lo stesso procedimento della somma anche sottrazioni: basta che un minuendo sia scritto come "complemento alla base".
Vediamo come si calcola una sottrazione utilizzando la base 2 ed il complemento a 2, per esempio:

$$\begin{array}{l} 27_{10} - \\ 12_{10} = \\ \hline 15_{10} \end{array}$$

Supponiamo di voler usare un sommatore a 8 bit (1 byte) per eseguire la sottrazione. Per prima cosa trasformiamo in base 2 i numeri:

$$27_{10} = 00011011_2$$
$$12_{10} = 00001100_2$$

Per passare da +12 a –12 in base 2 si deve calcolare il "complemento a 2" di 00001100_2, in due passi:

1) negare ogni bit del numero positivo: da 00001100_2 si ha 11110011_2
2) aggiungere 1: da 11110011_2 si ha 11110100_2

Cioè:

$$-12_{10} = 11110100_2$$

Eseguiamo ora la somma, trascurando il riporto oltre l'ottavo bit del risultato:

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1_2 + \\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0_2 = \\ \hline (1)\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1_2 \end{array}$$

Non tenendo conto del nono bit di riporto si ha: $1111_2 = 15_{10}$

1.7 L'INFORMAZIONE ANALOGICA E DIGITALE

L'informazione fornita da un segnale può essere discreta (e in questo caso assumere solo alcuni valori) o continua (può assumere una infinità di valori). In generale non è il carattere fisico del segnale che varia nei due casi, ma piuttosto lo scopo e gli obbiettivi del destinatario del segnale che fanno la differenza.

Esempio
Consideriamo un semaforo stradale con le sue luci: verde, giallo e rosso.

Un automobilista è interessato solo al colore presente quando sopraggiunge, per lui il semaforo è segnale con 4 valori: verde, giallo, rosso o spento. È sulla base di questa informazione che agirà di conseguenza, frenando se è il caso. Per l'automobilista il semaforo genera un segnale discreto.
Un addetto alla manutenzione è interessato alla quantità di luce emessa dal semaforo: la misurerà con un luxmetro, e sulla base della informazione ottenuta agirà di conseguenza, pulendo il vetro o cambiando la lampadina. Per l'addetto alla manutenzione il semaforo genera un segnale continuo.

Quando l'informazione è discreta è semplice digitalizzarla, cioè renderla numerica per poterla gestire con un computer. Il processo si chiama codifica: ad ogni valore si associa un numero, naturalmente in base 2. Ad esempio lo schermo del PC è composto da un elevato numero di punti (pixel) in funzione della risoluzione (800×600 = 480.000; 1024×768 = 786.432 …) ed il colore di ogni punto può essere codificato a sua volta in vari modi (16 bit, 32 bit, …). Questi valori si impostano come proprietà dello schermo e sia la scheda grafica che la stampante ed i programmi di grafica devono conoscere la codifica usata per funzionare correttamente.

Un altro esempio di codifica riguarda l'alfabeto: esiste ormai da molti anni lo standard ASCII (American Standard Code for Information Interchange) a 7 bit di codifica, per cui ad ogni carattere corrisponde un numero e viceversa ad ogni numero corrisponde uno e un solo carattere. Nella codifica ASCII sono inoltre presenti, nella parte iniziale, dei caratteri di controllo della stampante in quanto le sue origini sono legate ad aspetti di colloquio con le unità periferiche.

BIN	DEC	ASCII	BIN	DEC	ASCII	BIN	DEC	ASCII	BIN	DEC	ASCII
0	0	NUL	100000	32	SPACE	1000000	64	@	1100000	96	`
1	1	SOH	100001	33	!	1000001	65	A	1100001	97	a
10	2	STX	100010	34	"	1000010	66	B	1100010	98	b
11	3	ETX	100011	35	#	1000011	67	C	1100011	99	c
100	4	EOT	100100	36	$	1000100	68	D	1100100	100	d
101	5	ENQ	100101	37	%	1000101	69	E	1100101	101	e
110	6	ACK	100110	38	&	1000110	70	F	1100110	102	f
111	7	BEL	100111	39	'	1000111	71	G	1100111	103	g
1000	8	BS	101000	40	(	1001000	72	H	1101000	104	h
1001	9	HT	101001	41	)	1001001	73	I	1101001	105	i
1010	10	LF	101010	42	*	1001010	74	J	1101010	106	j
1011	11	VT	101011	43	+	1001011	75	K	1101011	107	k
1100	12	FF	101100	44	,	1001100	76	L	1101100	108	l
1101	13	CR	101101	45	-	1001101	77	M	1101101	109	m
1110	14	SO	101110	46	.	1001110	78	N	1101110	110	n
1111	15	SI	101111	47	/	1001111	79	O	1101111	111	o
10000	16	DLE	110000	48	0	1010000	80	P	1110000	112	p
10001	17	DC1	110001	49	1	1010001	81	Q	1110001	113	q
10010	18	DC2	110010	50	2	1010010	82	R	1110010	114	r
10011	19	DC3	110011	51	3	1010011	83	S	1110011	115	s
10100	20	DC4	110100	52	4	1010100	84	T	1110100	116	t
10101	21	NAK	110101	53	5	1010101	85	U	1110101	117	u
10110	22	SYN	110110	54	6	1010110	86	V	1110110	118	v
10111	23	ETB	110111	55	7	1010111	87	W	1110111	119	w
11000	24	CAN	111000	56	8	1011000	88	X	1111000	120	x
11001	25	EM	111001	57	9	1011001	89	Y	1111001	121	y
11010	26	SUB	111010	58	:	1011010	90	Z	1111010	122	z
11011	27	ESC	111011	59	;	1011011	91	[	1111011	123	{
11100	28	FS	111100	60	<	1011100	92	\	1111100	124	|
11101	29	GS	111101	61	=	1011101	93	]	1111101	125	}
11110	30	RS	111110	62	>	1011110	94	^	1111110	126	~
11111	31	US	111111	63	?	1011111	95	_	1111111	127	DEL

Tab. 1.17 - Elenco dei codici ASCII codifica a 7-bit

Con i caratteri ASCII si possono codificare testi in inglese, estendendolo a 8 bit (256 codici) si possono codificare i testi in quasi tutte le lingue "latine" europee (set *Latin-1*).

Attualmente si sta diffondendo una codifica più estesa che permette di gestire anche testi in lingue non latine (ebraico, arabo, cinese, giapponese, etc.) chiamato UNICODE, che utilizza caratteri di lunghezza variabile a 8, 16 bit, oppure fissa a 32 bit (rispettivamente UTF-8, UTF-16, UTF-32).

Anche in questo caso tastiera, schermo e stampante, nonché i programmi di gestione testi devono conoscere tutti la codifica usata ed essere in grado di gestirla correttamente.

Ad esempio Java utilizza caratteri a 16 bit in codice UNICODE (UTF-16). Per fortuna i primi caratteri UNICODE coincidono con i caratteri ASCII rendendo semplice la trasformazione.

Ma la maggior parte delle grandezze fisiche si presentano come continue, teoricamente con infiniti valori per ogni intervallo. In questo caso occorre misurarle con la precisione voluta, per ottenere come risultato un valore numerico (digitale) gestibile da un computer.

La precisione non è un fattore assoluto: dipende dagli obiettivi e dalle risorse (tecnologiche ed economiche) a disposizione.

Evidentemente una cosa è misurare la temperatura di una stanza per gestirne il riscaldamento e un'altra cosa è misurare la temperatura durante un esperimento di chimica, che richiederà una precisione ben maggiore.

Le cose si complicano quando la grandezza che ci interessa varia nel tempo o nello spazio. Dato che spazio e tempo sono anch'esse grandezze continue, qualunque loro intervallo contiene un insieme infinito di valori.

Esempio
Consideriamo il segnale audio: varia in modo continuo nel tempo. Riportando su un diagramma la variazione dell'intensità sonora nel tempo, otterremmo un andamento di questo tipo:

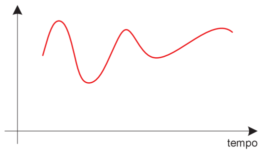


Fig. 1.11 – Segnale audio

L'ampiezza del segnale può assumere infiniti valori tra un massimo ed un minimo ed inoltre l'andamento è tondeggiante. La durata di questa forma d'onda è breve, tipicamente un millisecondo o due. Per digitalizzare questo tipo di segnali, occorre ripetere la misura più volte ed a breve intervallo, cioè *campionare* il segnale. L'idea è quella di approssimare la funzione analogica curvilinea con una funzione fatta a rettangoli.

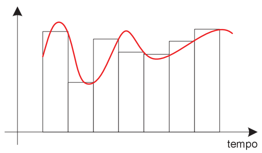


Fig. 1.12 – Campionamento grossolano

Ad intervalli di tempo regolari, si costruisce il rettangolo che approssima la funzione in quel punto. Se l'insieme dei rettangoli è sufficientemente fitto, la funzione di partenza è ben approssimata.

Naturalmente, se si aumenta la risoluzione, cioè il numero di rettangoli, l'errore di conversione diminuisce:

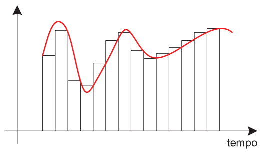


Fig. 1.13 – Campionamento più fine

Per quanto riguarda il valore dei singoli campioni si deve scegliere un certo grado di precisione. Praticamente si può dividere il campo di misura in intervalli uguali: più sono piccoli gli intervalli più la misura è precisa, ma comunque è arrotondata al valore dell'intervallo più vicino (*quantizzazione*).

La conversione di un segnale da analogico a digitale si realizza dunque con il campionamento e la successiva quantizzazione dell'ampiezza, ottenendo una sequenza discreta di valori.

Come è facile immaginare, sia la scelta dell'insieme finito di valori usato per rappresentare un singolo valore della grandezza, sia la scelta del numero dei campioni della grandezza in un intervallo continuo, sono aspetti cruciali rispetto al grado di attendibilità della rappresentazione digitalizzata dell'informazione.

Da una parte, minore è il numero di valori della griglia (numero dei campioni) peggiore è l'approssimazione della rappresentazione binaria; dall'altra, al crescere del numero di valori della griglia aumenta la dimensione della rappresentazione binaria dell'informazione, importante nelle applicazioni informatiche.

Occorre quindi trovare un compromesso tra le opposte esigenze di precisione e di compattezza della rappresentazione.

Esempio
I segnali audio musicali.

Le tracce audio digitali di un CD sono immagazzinate in file binari esattamente seguendo questo principio: ogni secondo vengono prelevati 44100 campioni e ogni campione è quantizzato con 16 bit. Considerando i due canali, sinistro e destro, la quantità di dati al secondo (*bit-rate*) è: 44100 × 16 × 2 = 1.411.200 bit/secondo = 176.400 Byte/secondo valore che corrisponde alla velocità di lettura "1X" dei normali lettori CD audio.

Tale bit-rate, chiamato anche "qualità CD", è diventato uno standard di riferimento quando si valutano i risultati prodotti dagli algoritmi di compressione.

Il valore di 44100 campioni/secondo deriva dal teorema del campionamento (o di Shannon) che impone una frequenza di campionamento almeno pari al doppio della massima frequenza presente nel segnale da campionare (nota: ogni segnale è scomponibile in una serie - di Fourier - di segnali di frequenze via via crescenti). Ma l'orecchio umano ha un limite di sensibilità pari a 20 KHz (oltre abbiamo gli ultrasuoni, percepiti da cani e pipistrelli, ma non dall'uomo). E' quindi inutile riprodurre musica o altri segnali audio per umani oltre i 20 KHz. Raddoppiando si ottiene 40 KHz, da cui la scelta di 44.100 campioni/secondo.

Lo standard **PCM**, utilizzato per gestire segnali audio in forma digitale non compressa, ha come standard proprio questi valori: 44100 come frequenza di campionamento e 16 bit per canale come quantizzazione.

Anche il formato **WAV** di Windows (Microsoft) è simile.

Un minuto di musica corrisponde a 176.400 × 60 = 10.584.000 Byte, cioè circa 10 MB per minuto! Un valore veramente elevato che ci fa intuire l'importanza delle tecniche di compressione (JPEG, MP3, etc.) per avere file musicali di dimensioni ragionevoli. Attenzione però: le tecniche di compressione più efficaci riducono il segnale originario (compressioni *lossy*), cercando nel contempo di mantenere invariata la sensazione uditiva dell'ascoltatore.

1.8 ESERCIZI

❒ 1.8.1

Semplificare le seguenti espressioni logiche:

a. $\overline{A}\cdot\overline{B}\cdot C\cdot D+\overline{A}\cdot\overline{\overline{B}}\cdot C\cdot D+A\cdot B\cdot C\cdot D$

b. $\overline{A}\cdot B\cdot C+\overline{A}\cdot B\cdot C\cdot D+\overline{\overline{A}\cdot B\cdot C}$

❒ 1.8.2

Dimostrare la validità o meno delle seguenti uguaglianze logiche:

a. $\overline{A}\cdot B+\overline{B}=\overline{B}$

b. $\overline{A}+\overline{B}\cdot B=\overline{B}+\overline{A}$

c. $\overline{A}+\overline{B}+B=1$

d. $\overline{A}\cdot\overline{B}+\overline{1}=\overline{B}\cdot\overline{A}$

e. $B+\overline{B}\cdot B=0$

❒ 1.8.3

Rappresentare le funzioni logiche F e G in termini delle variabili A e B, in forma normale e poi solo con operazioni NOR:

A	B	F	G
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

❒ 1.8.4

Rappresentare le funzioni logiche F e G in termini delle variabili logiche A, B e C, sia in forma normale che con numero di operazioni minimizzate:

A	B	C	F	G
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	1
1	1	1	1	0

❐ 1.8.5

Convertire i seguenti numeri decimali in binario, esadecimale ed ottale:

a. 37
b. 11
c. 255
d. 127
e. 1023

❐ 1.8.6

Se A e B rappresentano delle variabili logiche, descrivere per quali valori le seguenti eguaglianze sono soddisfatte:

a. $\overline{A} \cdot B = A \cdot \overline{B}$
b. $A \cdot B = A + B$