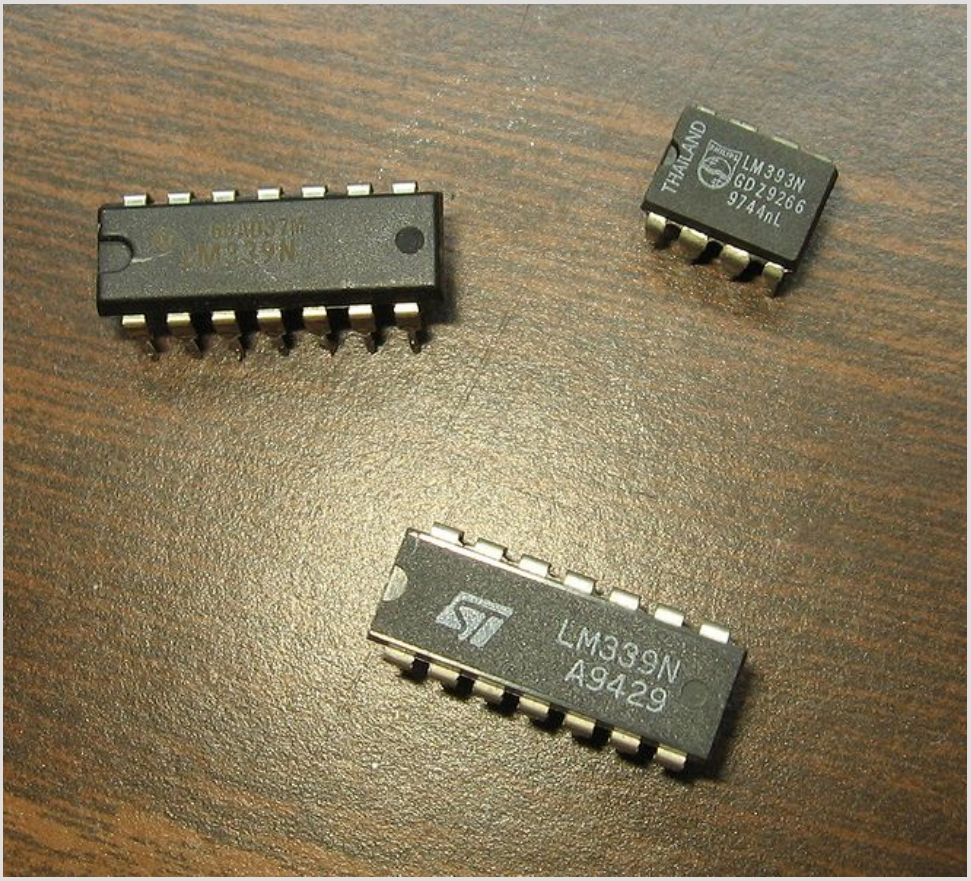
Alfonso Sommacal

ALGEBRE BOOLEANE E PROGETTO LOGICO DEI CALCOLATORI DIGITALI



Alfonso Sommacal

Algebre booleane e progetto logico dei calcolatori digitali

it.wikibooks.org
2021

Principale autore:
Alfonso Sommacal

Questa versione del libro è aggiornata al
3 ottobre 2021

In copertina:
Three comparator chips in DIP packages. *Autore:* Stuuf; *licenze:* CC BY-SA 3.0, GFDL; *fonte:* https://commons.wikimedia.org/wiki/File:Comparators_stuuf.jpg;

Indice

I Algebre booleane

Crediti

Parte I

Algebre booleane

1

Introduzione

1.1 I sistemi aritmetici

I calcolatori aritmetici moderni, le reti di comunicazione (come le centrali telefoniche), gli insiemi dei comandi automatici dei processi industriali (come quelli delle pile atomiche per esempio) sono, in totalità o in parte, dei sistemi aritmetici: essi hanno la caratteristica essenziale di manipolare delle informazioni discrete. In altre parole, i segnali che sono applicati al sistema (le entrate), o quelli che sono prodotti (uscite), prendono soltanto un numero finito di valori distinti (gli stati), che non variano con continuità.

Che i valori dei segnali siano discreti può essere un risultato della costruzione del sistema: i sassolini o le dita con cui si conta possono essere un esempio.

Ma il più delle volte è in seguito a una convenzione speciale che si quantificano delle grandezze fisiche che, macroscopicamente, ci appaiono invece come continue.

I calcolatori forniscono un esempio di questo processo di quantificazione.

I calcolatori in effetti si dividono in due categorie: *calcolatori analogici* e *calcolatori aritmetici (numerici o digitali)*.

1.1.1 Calcolatori analogici

Macchine calcolatrici in cui ogni grandezza matematica o fisica, su cui si vuole operare, viene rappresentata da una grandezza fisica variabile con continuità e nelle quali il calcolo viene eseguito costruendo un modello del sistema in esame.

Risolvono particolari tipi di problemi, simulandoli su modelli adeguati.

La principale applicazione dei calcolatori analogici si ha nello studio di sistemi descrivibili per mezzo di equazioni differenziali ordinarie ed equazioni differenziali alle derivate parziali.

Vengono altresì usati per la risoluzione di sistemi di equazioni algebriche lineari (simultanee) e per il calcolo delle radici di polinomi.

Il regolo calcolatore, inventato nel secolo XVII, è il primo esempio di calcolatore analogico sul quale, ad esempio, i logaritmi sono rappresentati da lunghezze a essi proporzionali (da cui il nome analogia).

Questo procedimento presenta però notevoli difficoltà se si lavora su dei fenomeni discontinui, e altresì presenta l'inconveniente che la precisione dei calcoli è limitata da quella degli organi utilizzati nel calcolatore.

I vantaggi più interessanti dei modelli analogici sono il tempo relativamente breve necessario per giungere alla soluzione del problema, e inoltre la possibilità, una volta introdotto il problema nell'elaboratore, di esaminare in pochissimo tempo le soluzioni corrispondenti a un vasto campo di valori dei parametri.

1.1.2 Calcolatori aritmetici (numerici o digitali)

All'analogia diretta utilizzata dal calcolatore analogico si sostituisce, nei calcolatori aritmetici, un'analogia indiretta.

Usufruendo della rappresentazione numerica delle grandezze, si applica l'operazione di codifica (analogia evocata prima) sulle cifre che compongono i numeri, in luogo di applicarla alle grandezze rappresentate da quest'ultimi.

Un calcolatore digitale rappresenta quindi numeri in forma codificata e questi assumono valori finiti o discreti.

Da un punto di vista strettamente concettuale, i calcolatori digitali traggono la loro origine dalla macchina descritta dal logico matematico Alan Mathison Turing nel 1936.

1.2 Storia del calcolatore

Il primo mezzo che si possa qualificare come un calcolatore digitale è l'abaco (il pallottoliere) ideato verso il 600 a.C.

L'abaco permette di rappresentare i numeri con la posizione delle palline o dischetti su una rastrelliera.

Le somme e le sottrazioni vengono effettuate rapidamente e con efficienza spostando in modo appropriato i dischetti (*abaculi*).

I problemi di navigazione, bellici, commerciali, che comportano la necessità di risolvere calcoli di notevole mole, suscitarono, a partire dal XVII, e nel XVIII secolo, un violento impulso per lo sviluppo delle "macchine" da calcolo.

Nel 1620 l'inglese Edmond Gunter ideava il regolo calcolatore. Allo stesso periodo risalgono le invenzioni di Blaise Pascal (1642) e di Gottfried Wilhelm von Leibniz (1651): il primo ideava una macchina addizionatrice, il secondo una moltiplicatrice, macchine funzionanti per mezzo di ruote azionate manualmente ma con riporto automatico.

Altro tentativo degno di essere menzionato è quello di Giovanni Poleni (1709) veneziano, che ideo una "macchina per calcolare", di cui realizzò un prototipo funzionante.

A Charles-Xavier Thomas de Colmar va il merito di aver costruito (1820) la prima calcolatrice venduta su scala commerciale.

Mentre un altro francese, Falcon, un secolo prima, aveva ideato il sistema delle schede perforate.

L'inglese Charles Babbage, che a diritto può essere considerato il padre del moderno calcolatore digitale, combinando le due idee su menzionate, inventava nel 1833 un dispositivo analitico e stabiliva un certo numero di principi fondamentali per la teoria dei calcolatori aritmetici.

Ma il moderno "computer" (calcolatore) è nato con lo sviluppo dei relais, dei tubi a vuoto e dei transistori: l'insieme dei dispositivi meccanici, magnetici ed elettrici di cui è costituito un calcolatore è chiamato correntemente "hardware".

Nel 1944 entra in funzione nell'università di Harward il Mark I^{o} a relais.

Nel 1946 è la volta dell'Eniac (Pensilvania University), a valvole.

Nel 1951 Univac e IBM, primi calcolatori a valvole in commercio.

Nel 1959 fanno la loro comparsa i calcolatori a transistor.

Nel 1960 nascono i calcolatori a circuiti integrati e i calcolatori a dispositivi superintegrati.

Oggi esistono calcolatori che compiono 20 milioni di operazioni al secondo, cioè una operazione ogni $50 \cdot 10\_9$ sec.

È continuata negli anni e continua con ritmo sempre più crescente la diffusione dei calcolatori sia nelle applicazioni di tipo organizzativo sia in quelle tecnico-scientifiche.

2

Sistemi di numerazione, aritmetica binaria

2.1 Struttura della memoria dal punto di vista operativo

In un calcolatore digitale la memoria è l'organo capace di conservare dei dati e delle istruzioni codificati in un certo modo. Si pone così il problema di cercare qual è il supporto migliore per memorizzare un'informazione. Dato un dispositivo che può assumere b stati diversi, è possibile codificare un numero in una certa base associando a ogni stato una cifra del numero in questione.

Per esempio, si pensi di avere un dispositivo capace di assumere 10 stati diversi ($S_0, S_1, S_2...S_9$) e di associare a ognuno di essi una delle dieci cifre base del sistema decimale nel seguente modo:

$$S_0 = 0 \quad S_1 = 1.....S_9 = 9$$

Ora se si vuole codificare il numero 3541 abbiamo bisogno di quattro dispositivi (perché quattro sono le cifre del numero da codificare) che chiameremo D_1, D_2, D_3 e D_4, dove

- D_4 assumerà lo stato corrispondente al numero delle unità
- D_3 a quello delle decine
- D_2 a quello delle centinaia
- D_1 a quello delle migliaia.

Stato	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
D_1				3						
D_2						5				
D_3					4					
D_4		1								

In definitiva per codificare un numero di **n** cifre in base **b** occorrono **n** dispositivi capaci di assumere **b** stati diversi, ed una volta costruito un tale sistema vi si possono codificare (beninteso non contemporaneamente) b^n numeri.

Vediamo ora come si può arrivare a dire che la base **2** è la più conveniente nel senso che a parità di numero di informazioni da memorizzare è necessario il minor numero di stati.

Si è visto che per memorizzare un numero di **n** cifre in base **b** occorre un dispositivo che contiene **b**×**n** stati diversi e capace di memorizzare $N = b_n$ informazioni diverse; il problema si pone quindi:

$$\mathbb{N} = b^n = cost \qquad b \cdot n = min$$

Immaginando ora di far variare **b** e **n** con continuità si hanno le condizioni estremali:

$$ndb + b \lg b \; dn = 0$$

$$\frac{\partial(b \cdot n)}{\partial b} db + \frac{\partial(b \cdot n)}{\partial n} dn = 0 \qquad ndb + bdn = 0$$

dalle quali segue che $lgb = 1$ cioè $b = e$ che si può verificare è effettivamente un punto di minimo per la funzione $b \times n$ con la condizione di $b^n = cost$.

Poiché $2 < e < 3$ si potrebbe scegliere tanto il 2 quanto il 3 come base ma per motivi tecnici (ad esempio quelli riguardanti il vantaggio dei dispositivi bistabili) si è scelta la base 2.

Vediamo ciò con un esempio pratico: il numero $999^{1}0 = 111101111_2$ (il numero in basso sta a indicare la base, il passaggio dalla base 10 alla base 2 verrà trattato in seguito). Allora volendo costruire i dispositivi atti a rappresentarli avremo:

Disp.	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
D_1										9
D_2										9
D_3										9

.....	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
S_0						0				
S_1	1	1	1	1		1	1	1	1	1

È facile vedere che nel primo caso occorrono 30 stati diversi, mentre nel secondo solo 20, e inoltre nel primo si possono memorizzare $10_3 = 1000$ informazioni mentre nel secondo $2^{1}0 = 1024$ informazioni.

Visto che la base 2 è migliore, vediamo come sono organizzati i dispositivi atti a memorizzare un numero in tale base.

Logicamente essi saranno in grado di assumere due stati diversi a cui diamo i valori: **1** o **0**; **+** o **-**; **vero** o **falso**.

I dispositivi vengono raggruppati (fisicamente) in insiemi, l'insieme delle informazioni binarie in essi contenute viene chiamato voce.

A ogni insieme viene associato un numero (indirizzo) che ci permette di accedere alla voce. Per esempio se si ha bisogno di conservare da qualche parte una informazione si dà al computer l'istruzione di memorizzarla nella voce, per esempio, 001.

2.2 Sistemi di numerazione

La nozione di numero è una delle nozioni fondamentali dell'aritmetica e della matematica. Non tracceremo qui né la storia di questa nozione né descriveremo l'evoluzione che ha portato alle definizioni assiomatiche dei numeri. Constateremo solamente

che in pratica i numeri ci risultano accessibili grazie a una rappresentazione che in generale è decimale, ma può anche essere di tipo differente (sessagesimale, romana...). Un metodo di rappresentazione costituisce quello che si chiama un sistema di numerazione. Il sistema più usato nella vita corrente è quello decimale.

Consideriamo il numero:

$$N_{10} = 15703$$

Questa notazione significa in pratica che **N** può considerarsi risultante dalla somma:

$$N = 1 \cdot 10^4 + 5 \cdot 10^3 + 7 \cdot 10^2 + 0 \cdot 10^1 + 3 \cdot 10^0$$

In altri termini, la scrittura ordinaria dei numeri è un modo di esprimerli in funzione delle cifre da **0** a **9** e delle potenze di **10**.

Questa notazione ha le seguenti proprietà:

- la posizione di una cifra indica la potenza di cui essa è coefficiente nel calcolo della somma
- in assenza di una potenza nella somma si mette uno **0** nella posizione corrispondente

non è necessario scrivere gli eventuali zeri a sinistra dell'ultima cifra non nulla.

A causa di queste proprietà si ha l'abitudine di dire che la numerazione decimale è una **numerazione di posizione**.

Il numero **10**, in funzione delle cui potenze si esprime il numero **N**, viene detto la base del sistema e sono necessari 10 simboli distinti 0, 1, 2, 3....9 per scrivere un numero.

L'utilizzazione della base 10 non è legata ad alcuna necessità logica o matematica ma probabilmente a considerazioni di tipo pratico; comunque si è fatto uso anche di altri sistemi di numerazione (base 12, 20, 60 in particolare). Si può allora generalizzare questa nozione di numerazione di posizione al modo seguente: dato un numero intero b>1, detto base del sistema del sistema di numerazione considerato, è b simboli distinti $x_0, x_1.....x_{b-1}$, si fa la convenzione di scrivere un numero intero sotto la forma:

$$X_b = x_n x_{n-1}....x_0$$

e questo per convenzione significa che il numero X può essere calcolato mediante la formula

$$X_b = x_n b^n + x_{n-1} b^{n-1} + ... + x_k b^k + ... + x_0 b^0$$

$$X_b = \sum_{k=0}^{k=n} x_k b^k$$

Ogni simbolo $x_n...x_k....x_0$ è uno fra i b simboli utilizzati per rappresentare i numeri da 0 a b-1.

Data la rappresentazione $x_n x_{n-1}.....x_0$ la sommatoria fornisce X univocamente, e inversamente si può dimostrare che la rappresentazione $x_n x_{n-1}....x_0$ di un intero è unica.

Nota 1:

- Per b=2 si ha un sistema di numerazione binario
- " b=8 ottale
- " b=10 decimale
- " b=12 dodecasimale
- " b=16 esadedcimale
- " b=60 sessagesimale

Nota 2: se si effettua la somma in un altro sistema di numerazione, il risultato sarà un numero X. Ad esempiosi può utilizzare un sisterma di posizione ma con una base **b** diversa.

Sarà allora

$$X' = X_{b'} = (x_n)_{b'}(b^n)_{b'}$$

Esempio: consideriamo il numero rappresentato in ottale da: $X_8 = 3017$.

Questo numero può essere rappresentato in decimale al modo seguente:

$$X_{10} = 3 \cdot 8^3 + 0 \cdot 8^2 + 1 \cdot 8^1 + 7 \cdot 8^0 = 1551$$

e quindi il numero che si scrive $X_8 = 3017$ in ottale diventa $X_10 = 1551$ in decimale.

Nota 3: l'indice k ha nome peso o rango: il peso più debole è 0, quello più forte è n.

Nota 4: è da notare la distinzione tra un numero e la sua rappresentazione; tale distinzione è importante e interviene nei problemi di conversione di un numero da un sistema di numerazione ad un altro.

Le definizioni precedenti si riferiscono a numeri interi. In generale si avranno espressioni comprendenti una parte intera e una frazionaria che scriveremo:

$$X_b = x -_n x_{n-1}...x_{-1}x_{-2}.....x_{-m}....$$

espressione del numero

$$X = \sum_{i=-}^{i=n} x_i b^i$$

Come nel sistema decimale la virgola separa la parte intera da quella frazionaria.

2.3 Sistema di numerazione binario

2.3.1 Numerazione binaria

In questo sistema un numero è rappresentato, conformemente alla definizione generale da una successione di 1 e di 0:

$$N_2 = x_n x_{n-1}....x_0, x_{-1}x_{-2}....x_m...$$

cioè

$$N_2 = \sum_{i=1}^{i=n} x_i 2^i$$

in cui i simboli x_i assumono i valori o **0** o **1** per ogni valore di **i** sono detti cifre binarie o *digits*.

Esempio:

$$N_2 = 1011,11$$

$$N_10 = 2^3 + 2^1 + 2^0, 2{-}1 + 2^{-2} = 11,75$$

Nota: con n cifre decimali è possibile rappresentare 10^n numeri. Per rappresentare questi numeri nel sistema binario è necessario un numero di cifre binarie **m** tali che

$$2^m > 10^n$$

cioè

$$m > 3,32 \cdot n$$

2.3.2 Addizione e sottrazione in binario. Complementazione

Addizione: consideriamo due numeri

$$X = x_n x_{n-1} .... x_i .... x_0$$

$$Y = y_n y_{n-1} .... y_i .... y_0$$

Si cerca $S = X + Y$. A tale scopo si procede come nel sistema decimale, partendo dal rango **0** e determinando rango per rango la somma **S**. Ad ogni rango **i** si determina:

- la somma modulo 2 di x_i, y_i e del riporto r_i ottenuto dal rango precedente, ottenendo il digit s_i della somma **S**.
- il riporto r_{i+1} da riportare sul rango **i+1**.

Queste due operazioni sono riassunte nella tabella 2.3.

X_i	Y_i	r_i	$X_i + Y_i + r_i$	S_i	r_i
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	1	1	0
0	1	1	10	0	1
1	0	0	1	1	0
1	0	1	10	0	1
1	1	0	10	0	1
1	1	1	11	1	1

Esempio:

$$R: \quad 011111100$$

$$X: \quad 101110110$$

$$Y: \quad 000001011$$

$$S: \quad 110000001$$

Sottrazione. In modo analogo si ha la tabella 2.4, che fornisce per ogni rango, la sottrazione S_i e il riporto r_{i+1} per la sottrazione **S=X-Y**.

X_i	Y_i	r_i	$X_i + Y_i + r_i$	S_i	r_i
0	0	0	0	0	0
0	0	1	$-1 = -2 + 1$	1	1
0	1	0	$-1 = -2 + 1$	1	1
0	1	1	-2	0	1
1	0	0	1	1	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	$-1 = -2 + 1$	1	1

Esempio:

$$\begin{aligned} R: &\quad 01011000 \\ X: &\quad 10110111 \\ Y: &\quad 01011110 \\ S: &\quad 01011001 \end{aligned}$$

In pratica però si utilizza il metodo dei complementi che riconduce il calcolo della sottrazione a quello di addizione.

Complementazione. Si definisce complemento vero di un numero intero binario $0 < x < 2^n$ la quantità:

$$C_{2^n}(X) = 2^n - X$$

Esempio: sia il numero X=11001 di cinque cifre. Il suo complemento vero (detto anche complemento a due) sarà:

$$C_{2^5}X = 100000 - 11001 = 111$$

Si definisce invece complemento a **1** di un numero intero binario $0 < X < 2'n$ la quantità:

$$C_1(X) = 2^n - X - 1$$

Esempio: sia **X**=11001. Il suo complemento a **1** sarà:

$$C_1(X) = 1000000 - 11001 = 110$$

Nel sistema binario il complemento a **1** di un numero può essere ottenuto direttamente scambiando nella rappresentazione del numero stesso gli **1** con gli **0**. Conoscendo il complemento a **1** di un numero binario si può ottenere il complemento vero sommando un **1**:

$$C_{2^n}(X) = C_1(X) + 1$$

Si può ora utilizzare la definizione di complemento e quella di classi residuo modulo **m** per costruire un algoritmo in grado di calcolare una sottrazione mediante un'operazione di somma.

Siano **A** e **B** due numeri tali che:

$$0 \leq A \leq 2^n \qquad 0 \leq B \leq 2^n$$

allora:

$$0 \leq |A - B| \leq 2^n$$

Si possono presentare due casi:
a) per $A - B \geq 0$ si ha:

$$A - B = |A - B|_{2^n} = |a - B + 2^n|_{2^n} = |A + (2^n - B)|_{2^n}$$

b) per $A - B \leq 0$ si ha:

$$A - B = 2^n - |A - B| = |A - B|_{2^n}$$

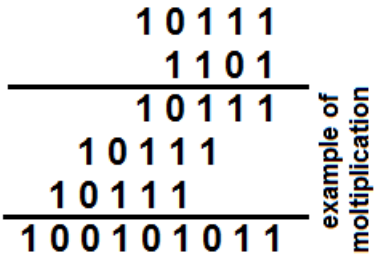
Per distinguere tra i due casi è sufficiente esaminare la posizione **n** nella operazione $A + C_2^n(B)$: infatti se $A + C_2^n(B) \geq 2^n$ nella n-esima posizione del risultato comparirà un **1** (caso a) e quindi il residuo modulo 2^n fornisce la differenza stessa; se invece compare lo **0** l'operazione fornisce il valore complementato del valore assoluto del risultato (caso b).

Esempio:

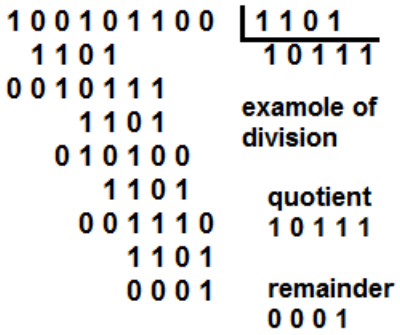
$$\begin{aligned} A = 111, \qquad & B = 10 \\ A - B = \quad & 111- \\ & \underline{....10} \\ & ..101 \end{aligned}$$

2.3.3 Moltiplicazione e divisione in binario

Moltiplicazione. Il metodo di base è quello che si usa anche in decimale: a seconda che la cifra del moltiplicatore valga **1** o **0** si aggiunge o no il moltiplicando alla somma parziale, scalando a ogni passo quest'ultima di un rango verso sinistra:



Divisione. Il metodo di base consiste nel sottrarre, come nel sistema decimale, il divisore moltiplicato per il solo fattore possibile e cioè **1**.



I metodi esposti per le quattro operazioni costituiscono solo dei procedimenti a partire dai quali sono stati trovati molti accorgimenti per accrescerne la velocità di esecuzione da parte del calcolatore.

2.3.4 Sistemi con base $2^{p(p\geq 1)}$

Per **p=1** si ha il binario puro. Per $p > 1$ si hanno dei sistemi (p=3 ottale, p=4 esadecimale) che vengono spesso utilizzati come notazioni abbreviate per il binario puro, sia per facilitare la lettura agli operatori, sia per certi calcoli. Le conversioni dalla base **2** alla base 2^p e viceversa si riconducono a dei semplici raggruppamenti di termini. Consideriamo dei numeri interri: pur di aggiungere il necessario numero di zeri a sinistra di un numero binario, si può sempre scrivere:

$$N = X_k X_{k-1}.....X_0$$

con

$$X_i = X_{8i+1)p-1}...X_{(i+1)p-1}.....X_{ip}$$
$$(i = 0, 1...k) \qquad (l = 1...p)$$

cioè raggruppando le cifre binarie a gruppi di **p** a partire da destra.

Segue che:

$$N = \sum_{i=0}^{i=k} X_i 2^{ip} = \sum_{i=0}^{(} i = k) X_i (2^p)^i$$

I gruppi X_i sono quindi in binario puro le rappresentazioni delle cifre di **N** nel sistema a base 2^p.

Inversamente, dato un numero **N** in un sistema di base 2^p, sarà sufficiente rappresentare ogni cifra in binario per avere il numero **N** in binario puro.

Esempio: codice ottale

$$\begin{array}{rrrr} N_2 = & 001 & 011 & 101 \\ N_8 = & ..1 & ...3 & ....5 \end{array}$$

Il codice ottale è quindi spesso utilizzato come notazione compatta del binario puro. Esso infatti, a causa della semplicità della procedura di conversione, permette di economizzare allora del tempo-macchina o di semplificare i dispositivi di conversione (caso di visualizzazione numerica a cifre luminose).

2.4 Conversioni da un sistema di numerazione ad un altro

2.4.1 Primo metodo

Numeri interi. Consideriamo un numero N; lo si può scrivere mediante due basi b_1 e b_2 ottenendo per **N** due espressioni che chiameremo N_b1 e N_b2. Si avrà allora:

$$N_{b_1} = x_n x_{n-1} .... x_0$$
$$N_{b_2} = y_m y_{m-1} .... y_0$$

con le relazioni:

$$N = \sum_{i=0}^{i=n} x_i b_1^n = \sum_{k=0}^{k=m} y_k b_2^k$$

Il problema della conversione è il seguente: data l'espressione N_{b_1} nel sistema di base b_1 trovare l'espressione N_{b_2} dello stesso numero **N** nel sistema di base b_2, o viceversa. A tale fine osserviamo che y_0 è il resto della divisione di **N** per la base b_2.

Infatti:

$$N = y_m b_2^m + y_{m-1} b_2^{m-1} + ... + y_1 b_2 + y_0$$

quindi:

$$(a) \qquad N = (y_m b_2^{m-1} + y_{m-1} b_2^{m-2} + ... + y_1) b_2 + y_0 \qquad (0 \leq y_0 \leq b_2)$$

cioè

$$N = N_1 b_2 + y_0 \qquad (0 \leq y_0 \leq b_2)$$

Possiamo fare le stesse considerazioni nel numero N_1 e quindi y_1 sarà il resto della divisione di N_1 per b_2 e così via.

Si vede quindi che si ottengono le cifre successive di N_2 considerando i resti successivi così ottenuti:

$$N = N_1 b_2 + y_0 \quad N_1 = N_2 b_2 + y_1 \quad N_2 = N_3 b_2 + y_2 .... N_m = N_{m+1} b_2 + y_m$$

in cui y_0, $y_1 ... y_m$ rappresentano i resti delle divisioni successive.

Esempio: convertire $N_1 0 = 3412$ in base **8**.

Si cercano allora i resti successivi delle divisioni per 8. Si ottiene:

$$\frac{3412}{8} \qquad resto = 4$$
$$.\frac{426}{8} \qquad resto = 2$$
$$..\frac{53}{8} \qquad resto = 5$$
$$...\frac{6}{8} \qquad resto = 6$$

e quindi

$$N_8 = 6524$$

Notiamo che il metodo di conversione tra due basi b_1 e b_2 sopra esposto è in teoria applicabile nei due sensi ($b_1 \to b_2 e b_2 \to b_1$). In pratica volendo passare dalla base b_1 alla base b_2, è necessario eseguire le operazioni di divisione nella base b_1. Quindi quando i calcoli sono fatti a mano non si ricorrerà a questo metodo che per convertire un numero decimale in un'altra base, come nell'esempio visto. Lo stesso discorso non vale nel caso in cui l'operazione di conversione sia meccanizzata mediante un sistema automatico.

Numeri frazionari inferiori all'unità. Consideriamo, analogamente al caso precedente, in numero **N** in una rappresentazione N_{b_1} in un sistema di base b_1; e cerchiamo la rappresentazione N_{b_2} in un sistema di base b_2. Notiamo che l'espressione:

$$N = (0, y_{-1}y_{-2}....y_{-m}.....)b_2$$

significa:

$$N = y_1 \cdot b_2^{-1} + y_{-2} \cdot b_2^{-2} + ... + y_m \cdot b_2^{-m} + ...$$

Moltiplicando **N** per b_2 si ha:

$$(b) \qquad (N \cdot b_2) = y_{-1} + y_{-2} + ... + y_{-m}$$

Cioè la cifra che in $N \times b_2$ appare a sinistra della virgola è la prima cifra cercata: y_1. Possiamo ora considerare il numero ottenuto conservando solo le cifre a destra della virgola ed applicare lo stesso procedimento: cioè lo si moltiplica per b_2 e la cifra che risulta a sinistra della virgola sarà y_2, ecc.

Il procedimento considerato può anche non avere termine, e quindi la rappresentazione di un numero frazionario può avere termine in un sistema di numerazione e non averlo in un altro.

Come per la conversione di numeri interi, si può notare che il metodo qui visto per i numeri frazionari comporta delle moltiplicazioni nel sistema di partenza a base b_1. Quindi, per i calcoli a mano, questo metodo si utilizza solamente per le conversioni a partire dal sistema decimale verso una generica base **b**.

Esempio: Convertire $N_10 = 0,71875$ in base 2.

		$..0,71875 \cdot 2$
parte intera	$1+$	$0,43750 \cdot 2$
parte intera	$0+$	$0,87500 \cdot 2$
parte intera	$1+$	$0,75000 \cdot 2$
parte intera	$1+$	$0,50000 \cdot 2$
parte intera	$1+$	$0,00000 \cdot 2$
parte intera	$0+$	$0,00000 \cdot 2$

e quindi $N_2 = 0,101110$ ha termine dopo la 5_a cifra.

Numeri con parte intera e parte frazionaria. Si procede in due tempi: si applica il metodo (a) alla parte intera e il metodo (b) alla parte frazionaria. Il numero cercato si ottiene mediante unione dei due risultati.

2.4.2 Secondo metodo

Dato un numero avente la rappresentazione:

$$N_{b_1} = x_n x_{n-1} .... x_0, x_{-1} x_{-2} .... x_{-k}$$

si esprimono le cifre x_n, $x_{n-1}, ..., x_0, x_{-1}$ nel sistemadi numerazione b_2 e così anche per le potenze di b_1: b_1^n $b_1^($$n-1)...b_1, b_1^($$-1)$ e si effettua la somma seguente nel sistema di base b_2:

$$(c) \qquad N_{b_2} = \sum_{-inf.}^{n} (x_i)_{b_2} \cdot (b_1^i)_{b_2}$$

In questo caso notiamo che il passaggio dalla base b_1 alla base b_2 implica che la somma (c) sia effettuata nel sistema di base b_2, e quindi questo metodo è quello usato quando è necessario convertire un numero da una base generica a quella decimale nei calcoli fatti a mano.

Esempio: convertire $N_2 = 11011,101$ in base 10.

$$N_{10} = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} =$$
$$= 16 + 8 + 2 + 1 + 0,5 + 0,125 = 27,625$$
$$N_{10} = 27,625$$

I metodi di conversione esposti si prestano a numerose varianti: i parametri da prendere in considerazione dal punto di vista tecnico consistono essenzialmente nella rapidità di conversione e nella complessità dei dispositivi, in contrasto fra loro.

Fonte del testo:
https://it.wikibooks.org/w/index.php?title=Algebre_booleane_e_progetto_logico_dei_calcolatori_digitali/Sistemi_di_numerazione,_aritmetica_binaria&oldid=349855

3

Codici

3.1 Generalità

I calcolatori elettronici, per motivi di realizzazione, utilizzano una rappresentazione interna sia delle grandezze sia dei simboli di tipo binario. A ogni numero o simbolo viene fatta corrispondere una parola binaria, cioè una successione di **0** e **1** definendo quindi un codice binario.

In generale, stabilire una corrispondenza biunivoca tra i simboli di un insieme E_1 (insieme dei messaggi da codificare) e quelli di un insieme E_2 (insieme delle parole-codice) è per definizione *codificare* gli elementi di E_1 mediante gli elementi di E_2.

In pratica si dice *codice* la corrispondenza (cioè l'insieme delle regole di traduzione) tra gli insiemi E_1 e E_2. Il seguito del capitolo tratta i codici (essenzialmente binari) utilizzati per la rappresentazione delle informazioni. Seguirà una descrizione dei metodi per l'addizione, la sottrazione, la moltiplicazione e la divisione decimale.

Verrà inoltre trattata l'aritmetica floating-point, i cheks di parità e i codici autocorrettori.

3.1.1 Codice binario puro

Il sistema di numerazione binario, descritto nei paragrafi precedenti, costituisce uno dei procedimenti mediante i quali si può rappresentare un numero sotto forma binaria.

Si è visto però che le conversioni binario-decimali richiedono un procedimento relativamente complesse e quindi, per evitare o semplificare il più possibile questa operazione di conversione tra dati esterni e rappresentazioni interne, si ricorre in certi casi a un altro tipo di codifica binaria fornita dai codici di tipo *B.C.D.* (binary-coded-decimal).

3.1.2 Codici di tipo B.C.D.

Il principio consiste nel codificare il numero rappresentato in base decimale, cifra decimale per cifra decimale ottenendo la rappresentazione binaria cercata accostando la codifica binaria delle varie cifre. La tabella **3.1** fornisce un esempio di codifica delle cifre decimali: si utilizza semplicemente la loro rappresentazione in binario puro prendendo il numero minimo di simboli binari, e cioè **4**.

Tabella 3.1: Un codice B.C.D.

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	1	1	1	0	0
0	0	1	1	0	0	1	1	0	0
0	1	0	1	0	1	0	1	0	1

Mediante questo codice il numero decimale 1358 sarà rappresentato da:

$$0001 \quad 0011 \quad 0101 \quad 1000$$

Per codificare in binario le 10 cifre da 0 a 9 sono necessarie almeno quattro posizioni binarie, che permettono di fornire $2^4 = 16$ combinazioni. Se si considera quindi un codice a quattro posizioni, si può definire un codice particolare scegliendo 10 combinazioni tra le sedici permesse per attribuirle alle cifre 0.1.2..9, in un certo ordine. Si ha che il numero dei codici di 10 cifre decimali mediante 4 posizioni binarie è dato dal numero di disposizioni di classe 10 di 16 oggetti, cioè:

$$D = \frac{16!}{6!} = 2.910^{1}0$$

In pratica però non tutti i codici così formati sono effettivamente distinti, nel senso che non consideriamo distinti due codici i quali differiscano solamente per la permutazione delle variabili, e nel circuito a uno scambio di fili.

Inoltre, nel caso che tutte le variabili siano disponibili sotto le due forme complementata e non complementata (uscita dei flip-flop, per esempio) possono ancora essere considerati equivalenti due codici che differiscono solo per la complementazione di una o più delle quattro posizioni.

Comunque, malgrado ciò, il numero dei codici distinti resta dell'ordine dei milioni.

3.1.3 Codici B.C.D. pesati a 4 posizioni

Il codice utilizzato in precedenza (tabella 3.1) e cioè quello binario puro, è di tipo pesato. Se si pone:

$$X = x_3 x_2 x_1 x_0 \tag{3.1}$$

per la cifra decimale, x_0 x_1 x_2 x_3 simboli binari della corrispondente parola-codice, si ha:

$$X = x_3 2^3 + x_2 2^2 x_1 2^1 x_0 2^0 = \sum_{i=0}^{i=3} x_i 2^i$$

$$X = 8x_3 + 4x_2 + 2x_1 + x_0 \tag{3.2}$$

Le cifre 8, 4, 2, 1 costituiscono i pesi (cioè i coefficienti da attribuire a x_3, x_2, x_1, x_0 quando si ricostruisce X mediante la (1)). Il codice della tabella 3.1, detto codice (8,4,2,1) non è che un esempio fra i tanti di codice pesato a 4 posizioni. In un tale codice, ogni cifra decimale da 0 a 9 è rappresentata da una espressione del tipo

$$\sum_{i=0}^{i=3} x_i J_{P_i} \qquad (x_i = 0 \quad oppure \quad 1; i = 0, 1, 2, 3; J = 0.1..9.$$

I coefficienti P_i costituiscono i **pesi** caratteristici del codice considerato e si usa indicare il codice mediante la successione ordinata dei P_i: si avrà quindi il codice 8421, il codice 2421, ecc. Si hanno nella tabella 3.2 cinque esempi di codici a pesi positivi.

Tabella 3.2: Cinque codici B.C.D. a pesi positivi

	8421	7421	2421	4311	5121
0	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0001
2	0010	0010	0010	0011	0010
3	0011	0011	0011	0100	0011
4	0100	0100	0100	1000	0111
5	0101	0101	1011	0111	1000
6	0110	0110	1100	1011	1001
7	0111	1000	1101	1100	1010
8	1000	1001	1110	1110	1011
9	1001	1010	1111	1111	1111

Notiamo che nel codice pesato (a quattro o più posizioni) ogni gruppo decimale codificato in binario può essere facilmente convertito in un segnale analogico a dieci livelli: sommando correnti proporzionali ai pesi si ottiene una corrente proporzionale alla cifra decimale considerata.

3.1.4 Codici B.C.D. non pesati a quattro posizioni

Non tutti i codici sono pesati. Tra i codici non pesati a quattro posizioni, quello che viene più usato è il codice *a eccesso di 3*.

In questa rappresentazione, alla cifra decimale X ($0 \leq x \leq 9$) si fa corrispondere la rappresentazione binaria pura di X+3 (tabella 3.3)

Tabella 3.3: Codice non pesato ad accesso di 3

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1
0	1	1	1	1	0	0	0	0	1
1	0	0	1	1	0	0	1	1	0
1	0	1	0	1	0	1	0	1	0

Questo tipo di codice è di lettura meno immediata rispetto al codice 8421 per un operatore umano e si presta meno bene alla conversione numerica analogica delle cifre decimali: ma dal punto di vista della realizzazione dei circuiti aritmetici esso possiede notevoli proprietà di simmetria: in particolare il fatto di essere *autocomplementare*. Infatti per ogni $X = (x_3 x_2 x_1 x_0)$ si ha

$$C_9(x) = 9 - x = (1 - x_3), (1 - x_2), (1 - x_1), (1 - x_0)$$

ove $C_9(x)$ è il complemento a 9 di x: per ottenere il complemento a 9 di una cifra è sufficiente complementare in binario, cifra dopo cifra, la rappresentazione $x_3x_2x_1x_0$ di X. Nel caso del codice 8421 ad es., le cifre binarie di **9-x** non dipendono in maniera così diretta da quelle di X.

3.1.5 Codici B.C.D. a più di quattro posizioni

Codice *2 su 5*. Per questi tipi di codici B.C.D., si utilizzano 5 posizioni binarie per rappresentare le 10 cifre da 0 a 9.

A tale scopo si scelgono le 10 combinazioni binarie che hanno 2 cifre su cinque uguali a 1.

La tabella 3.4 dà un esempio di codice di questo tipo, corrispondente a una permutazione delle dieci parole binarie in oggetto.

Questo non è un codice pesato. La nozione di codice pesato si generalizza facilmente al caso di qualunque numero di cifre binarie. Inoltre, affinché un codice sia autocomplementare, è necessario (ma non sufficiente) che la somma dei pesi sia uguale a 9. Anche la nozione di codice ad eccesso di 3 può essere esteso a codici di eccesso di E.

3.2 Codici a distanza unitaria

3.2.1 Generalità (caso binario)

Si dice codice a distanza unitaria un codice binario che possiede la seguente proprietà: le rappresentazioni in questo codice di due numeri consecutivi differiscono per una sola posizione. Se si definisce distanza di Hammimg D(X,Y) fra le due parole-codice X e Y il numero di posizioni differenti, si ha che dati due numeri x_{i+1} e x_i aventi per parole codice X_{i+1}, X_i e tali che $x_{i+1} - x_i = 1$, questa proprietà è espressa dalla relazione

$$[(x_{i+1} - x_i) = 1] \quad \Rightarrow \quad [D(X_i, X_{i+1}) = 1]$$

x	X
7	0111
8	1000

Si ha:

$$D(X_7, X_8) = 4$$

3.2.2 Codice riflesso (Codice Gray)

Costruzione di un codice riflesso a **n** ranghi binari. Questo codice è rappresentato (per 4 posizioni) nella tabella 3.5 e può essere costruito come segue:

Considerate il blocco verticale (b_1) a una colonna e due righe. Scrivete al di sotto un blocco (b'_1) simmetrico rispetto all'asse b_1/b'_1. Per il rango binario a sinistra (rango 1) porre degli zeri di fronte al blocco (b_1) e degli **1** di fronte al blocco (b'_1). In tal modo si ottiene un blocco (b_2) a 2 colonne e 2^2 righe.

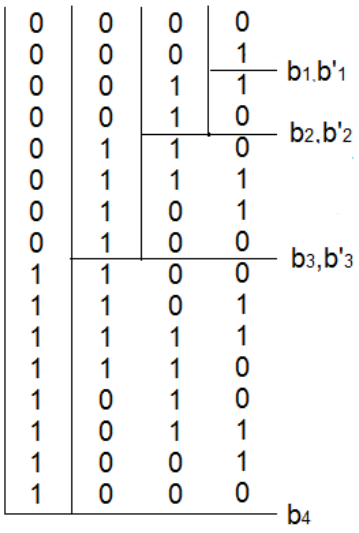
Scrivete al di sotto di (b_2) un blocco (b'_2) simmetrico e completate il rango a sinistra (rango 2) con degli **0** di fronte a (b_2) e degli **1** di fronte a (b'_2). Si ottiene così un blocco (b_3) a 3 colonne e 2^3 righe.

Ripetere l'operazione di volta in volta, ecc.

Questo codice è un codice a distanza unitaria. Infatti, se un blocco (b_n) è ottenuto in tale modo e possiede tale proprietà, il blocco $(b_{(n+1)})$ la possiede anch'esso, perché: la cifra di sinistra cambia solo attorno all'asse di simmetria **n** mentre il resto della parola (parte destra) non cambia.

Là ove non cambia la cifra di sinistra si hanno solo i cambiamenti propri del blocco (b_n) che è per ipotesi un codice a distanza unitaria.

Quindi (b_n) possiede la proprietà di distanza unitaria essendo costruito a partire da (b_1) per cui tale proprietà è verificata.



Formule di conversione Binario puro-Binario riflesso: si verifica facilmente che le cifre B_k in binario puro sono ottenute dalle cifre R_k in binario riflesso mediante la relazione:

$$(1) \qquad B_k = |\sum_{i=k}^{i=n} R_i|_2 \qquad per.ogni.k$$

mentre inversamente si ha:

$$R_k = |B_{k+1} + B_k|_2$$

Esempio: conversione del Binario puro 10110.

Con invarianza del valore scriviamo 010110, e operiamo come segue:

R4 = B5 XOR B4 = 0 XOR 1 = 1
R3 = B4 XOR B3 = 1 XOR 0 = 1
R2 = B3 XOR B2 = 0 XOR 1 = 1
R1 = B2 XOR B1 = 1 XOR 1 = 0
R0 = B1 XOR B0 = 1 XOR 0 = 1

ottenendo 11101 come conversione di 10110.

Queste formule di conversione mostrano che il calcolo dei simboli di rango **k** fa intervenire le quantità di rango superiore, ciò implica che la conversione ha inizio dai pesi maggiori. Questo costituisce un inconveniente in pratica in quanto negli organi di calcolo in binario puro, a causa della struttura del codice binario, le operazioni il più delle volte vengono fatte a partire dai pesi più piccoli.

La formula (1) si può però mettere sotto una forma che si presta a un trattamento per ranghi crescenti:

$$B_0 = |\sum_{i=0}^{i=n} R_i|_2$$

$$B_1 = |\sum_{i=1}^{i=n} R_i|_2 = ||\sum_{i=0}^{i=n} R_i|_2 + R_0|_2$$

$$B_2 = ||\sum_{i=0}^{i=n} R_i|_2 + |R_0 + R_1|_2|_2$$

.....

$$B_k = ||\sum_{i=0}^{i=n} R_i|_2 + |\sum_{i=0}^{i=k-1} R_i|_2|_2$$

Nel caso che l'operazione di conversione sia eseguita con un procedimento in serie, questa formula da luogo a una semplice realizzazione: è sufficiente infatti far circolare due volte la parola in codice riflesso in un **flip-flop** di tipo **T** al cui ingresso arrivano successivamente le cifre: in uscita del flip-flop di tipo **T** (inizialmente a zero) si avrà la somma modulo 2 dei valori d'ingresso. La prima circolazione permette la determinazione del valore B_0, mentre durante la seconda circolazione il **flip-flop** darà come uscite successive i simboli B_k.

3.2.3 Utilità dei codici a distanza unitaria

I codici a distanza unitaria sono molto utili ogni qualvolta si vogliono evitare transizioni contemporaneamente su più posizioni. Consideriamo, ad esempio, il passaggio da **0111** a **1000** e cioè fra cifre che pur differendo di una sola unità, comportano la variazione di tutte e quattro le posizioni della rappresentazione binaria. Il passaggio, percome sono costruiti i codificatori, non avviene simultaneamente, ma attraverso gli stati scaglionati:

$$0111 \longrightarrow 0011 \longrightarrow 0001 \longrightarrow 0000 \longrightarrow 1000$$

Quindi gli errori, riferiti al valore finale **1000** sono rispettivamente per ogni stadio pari a 1,5,7,8,0. I codici a distanza unitaria non sono quindi soggetti a questo tipo di inconveniente.

3.3 Controllo degli errori

I circuiti di commutazione elettronici e in particolare quelli dei calcolatori digitali non sono dotati di perfetto grado di affidabilità: si possono generare errori dovuti a cause diverse quali elementi parassiti, perdite di alimentazione, errori casuali dati da componenti oscillanti al di fuori della tolleranza, errori sistematici dovuti al cattivo funzionamento di un componente, per usura o per guasto.

4

Teoria della commutazione

4.1 Relais elettromagnetici con interruttori

Numerosi sistemi elettrici hanno nei loro circuiti di comando e di controllo dei relais elettromagnetici. Un tale relais è composto essenzialmente da un elettromagnete e da una armatura mobile, portante delle lamine metalliche.

Ogni lamina è fissata all'armatura da una estremità mentre l'altra estremità è libera e può, a seconda della posizione dell'armatura, chiudere o no un contatto. L'insieme della lamina e del punto di contatto associato, costituisce ciò che si chiama interruttore: esso può essere aperto o chiuso. A seconda che la bobina di eccitazione della elettrocalamita sia eccitata o no da una corrente, l'armatura portante si trova in una o nell'altra di due posizioni: quando non passa corrente d'eccitazione, essa si trova nella posizione detta di **riposo** mentre de la bobina è eccitata, l'armatura si trova in una seconda posizione.

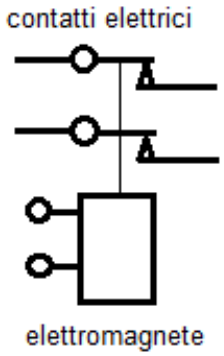
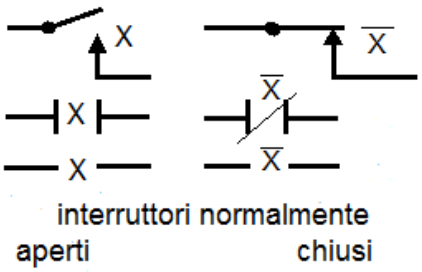


Fig. 4.1: Relè



Si può allora distinguere nella maggior parte dei relè due tipi principali di interruttori: interruttori normalmente aperti e interruttori normalmente chiusi. Gli interruttori normalmente aperti sono aperti quando l'armatura è nella posizione di riposo e chiusi quando questa è attirata dalla elettrocalamita. Gli interruttori normalmente chiusi sono chiusi quando l'armatura è nella posizione di riposo e aperti quando essa è attirata dalla elettrocalamita.

Riassumendo quindi, il relè descritto possiede le caratteristiche seguenti:

1. c'è un valore di ingresso costituito dallo stato della bobina di eccitazione: essa può essere eccitata o no.

2. vi è un certo numero di uscite, ciascuna delle quali è costituita dallo stato di chiusura o di apertura degli interruttori.

Il rapporto ingresso-uscita (cioè la relazione tramite la quale il relè apre o chiude un contatto in funzione del valore in ingresso) dipende dalla natura degli interruttori: essa può essere riassunta nella tabella seguente:

4.2 Variazioni binarie associate a un relè. Complemento di una variabile

L'esame della tabella rivela che il comportamento di un relè può essere descritto mediante delle variabili che possono assumere 2 valori, cioè delle variabili binarie.

È comodo fare astrazione dal nome delle variabili e utilizzare al posto delle coppie di simboli A_0, A_1, ecc., 2 simboli solamente (gli stessi per tutte le variabili). Si possono prendere per questo le cifre 0 e 1 e attribuirle in maniera arbitraria ai due stati di ogni variabile:

Si può osservare che quando la variabile X ha un dato valore, la variabile Y ha l'altro. Si può allora porre:

$$Y = \bar{X}$$

dove la variabile Y è definita in funzione di X: essa si chiama il *complemento di* X:

Tabella 4.1: Complemento di X

X	$\bar{X}$
0	1
1	0

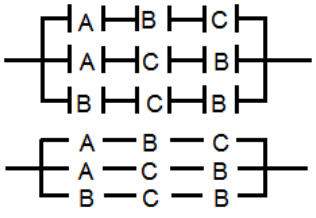
Si hanno allora le relazioni:

$$\begin{cases} A = E \\ X = A \\ Y = \bar{X} = \bar{A} \end{cases}$$

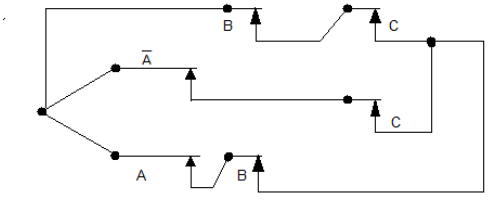
Si vede quindi che con questa convenzione, la variabile X associata ad un interruttore normalmente aperto è uguale ad E, mentre quella associata ad un interruttore normalmente chiuso è $\bar{E}$, per cui è sufficiente una sola variabile E a definire il relè.

Nasce quindi la convenzione di associare ad ogni relè una lettera X. Il valore della variabile rappresenta lo stato di eccitazione del relè e si indicano con X gli interruttori *normalmente aperti* e con $\bar{X}$ gli interruttori *normalmente chiusi*.

Esempio: tre rappresentazioni di un circuito con interruttori:



circuiti con interruttori



circuito con interruttori

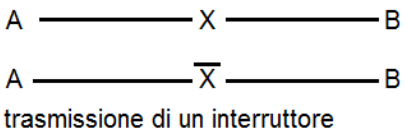
Le differenti lettere che compaiono in uno schema corrispondono ai diversi relè, i differenti simboli X e $\bar{X}$ corrispondono agli interruttori di un relè e la presenza o l'assenza della sbarra sulla lettera informa della natura dell'interruttore (normalmente aperto o normalmente chiuso).

4.3 Funzione di trasmissione di un circuito

In generale, dato un circuito a 2 terminali A e B, si definisce una variabile binaria T_{AB} chiamata **funzione di trasmissione** tramite la seguente convenzione:

$$\begin{cases} T_{AB} = 1, & \text{il } \textit{cammino} \text{ AB è chiuso} \\ T_{AB} = 0, & \text{il } \textit{cammino} \text{ AB è aperto} \end{cases}$$

(a) *Trasmissione di un interruttore*
Dato un circuito costituito da un unico interruttore, si hanno le seguenti relazioni



trasmissione di un interruttore

Interruttore normalmente aperto:

$$\begin{cases} T = 1, & \text{se X=1} \\ T = 0, & \text{se X=0} \end{cases}$$

che permette di scrivere algebricamente:

$$T = X$$

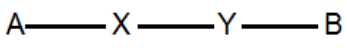
Interruttore normalmente chiuso:

$$\begin{cases} T = 1, & \text{se X=0} \\ T = 0, & \text{se X=1} \end{cases}$$

che permette di scrivere algebricamente:

$$T = \bar{X}$$

(b) *Trasmissione in un circuito*
Si consideri il circuito costituito da 2 interruttori X e Y in serie (e la cui natura è generica):



trasmissione in un circuito

Se gli interruttori X e Y sono entrambi chiusi, il circuito è chiuso. Viceversa, se l'uno dei due interruttori è aperto, o se lo sono entrambi, il cammino tra A e B è aperto.

Si può rappresentare tutto ciò tramite una tabella (tavola della verità) che ha per ingresso le differenti combinazioni possibili di X e Y e, come uscite, il valore della trasmissione T_AB del circuito tra A e B. Così la risposta binaria del circuito appare come una funzione $T = T(X, Y)$ delle due variabili binarie X e Yo, ancora, come una operazione sulle variabili X ed Y.

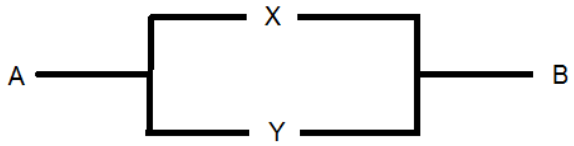
Si porrà allora:

$$T = X * Y = XY$$

Così all'operazione di messa in serie dei 2 interruttori X ed Y è associata l'operazione (*) sui valori algebrici delle variabili X ed Y. Tale operazione si chiama *prodotto*.

(c) *Trasmissione di un circuito con 2 interruttori in parallelo*

Si consideri il circuito costituito da 2 interruttori X e Y (di natura qualsiasi) posti in parallelo. Se uno dei due interruttori X o Y è chiuso, o se sono entrambi chiusi, il cammino tra A e B è chiuso. Viceversa se i due interruttori sono aperti contemporaneamente, il cammino AB è aperto.



Il funzionamento del circuito AB potrà essere riassunto:

La risposta T_{BA} del circuito appare come una funzione delle due variabili X e Y oppure come una operazione su X e Y che si scriverà:

$$T = X + Y$$

Così all'operazione di messa in parallelo di 2 interruttori è associata l'operazione (+) sulle risposte di X e Y, ed è chiamata somma.

Sono così definite tre operazioni sugli interruttori.

1. La *complementazione*: essa consiste nel sostituire ad un interruttore in parallelo X un interruttore $\bar{X}$ dell'altro tipo: la nuova trasmissione è $T = \bar{X}$.
2. La messa in serie: la trasmissione risultante è $T = X * Y$
3. La messa in parallelo: la trasmissione risultante è $x * y$.

Finora si sono considerati dei circuiti costituiti da interruttori elementari, e mediante le tre suddette operazioni si sono costruiti altri circuiti, ma si possono naturalmente definire le operazioni -, +, * anche per dei generici circuiti.

- *Complementazione*: si dice circuito complementare di un circuito di trasmissione T, un circuito che da la risposta $T_c = \bar{T}$
- *Messa in serie*: $T_1 * T_2$ dove T_1 e T_2 sono le risposte dei due circuiti
- *Messa in parallelo*: $T_p = T_1 + T_2$

Quindi con le operazioni di complementazione, messa in serie e messa in parallelo si può definire un procedimento di costruzione di circuiti sempre più complessi partendo da un circuito elementare. Ad ogni passo costituito da una operazione sui circuiti, si può associare una operazione algebrica sulla risposta dei circuiti. Al circuito finale si potrà quindi associare una risposta algebrica ed esprimerla in funzione di quella degli interruttori dei differenti relè.

Prima di dare qualche esempio di questo metodo, si esporranno alcune proprietà delle operazioni di -, *, +.

Per dimostrare queste proprietà si può procedere in due maniere:

1. ragionando direttamente sui circuiti
2. utilizzando le tavole di definizione delle operazioni.

4.4 Proprietà delle operazioni (-), (*), (+)

Prima di passare alla trattazione delle operazioni (-), (*), (+), diamo qui di seguito la definizione di circuiti equivalenti.

Due circuiti si dicono equivalenti, se le variabili associate ai circuiti assumono gli stessi valori, ogni qualvolta gli interruttori, comuni ai due circuiti, si trovano nello stesso stato.

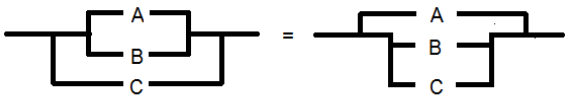
Due circuiti possono essere equivalenti anche se non tutti gli interruttori, di cui sono formati, sono in comune.

In questo caso però le variabili associate ai circuiti non devono dipendere da questi interruttori.

Si indicano le proprietà delle operazioni (-), (*), (+) con i simboli da R_1 a R_20.

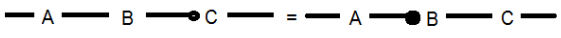
Si lascia al lettore, per esercizio, la dimostrazione di tali relazioni.

R_1)



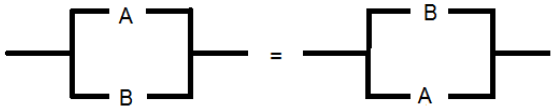
(A+B)+C=A+(B+C) [proprietà associativa

R_2)



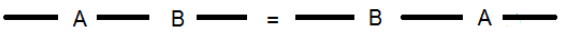
(A*B)*C=A*(B*C) [proprietà associativa]

R_3)



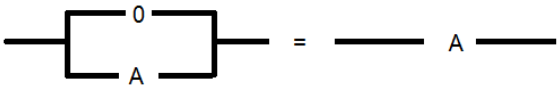
A+B=B+A [proprietà commutativa]

R_4)



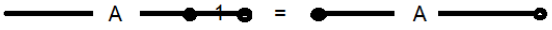
A*B=B*A (proprietà commutativa del prodotto)

R_5)



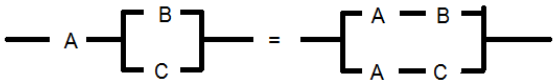
A+0=0+A=A (teorema della unicità dello 0)

R_6)



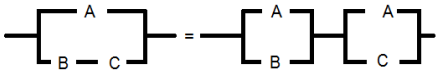
A*1=1*A=A (1 elemento neutro della moltiplicazione)

R_7)



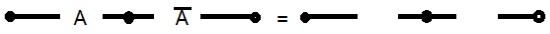
A*(B+C) (proprietà distributiva del prodotto)

R_8)



A+B*C=(A+B)*(A+C) (Proprietà distributiva della somma sul prodotto)

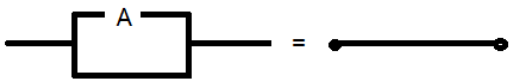
R_9)



$A*\overline{A}=0$

(Il prodotto di una variabile per il suo complemento è = 0)

R_{13})



A+1=1 (esistenza di massimo)

A titolo di esempio diamo qui di seguito la dimostrazione della relazione

$$A + B * C = (A + B) * (A + C)$$

Primo metodo: ragionando sui circuiti si vede che se A è aperto, i due circuiti sono chiusi solamente se B e C sono chiusi, e aperti negli altri casi.

Se A è chiuso, i due circuiti sono chiusi qualunque sia lo stato di B e C. I due circuiti hanno quindi lo stesso funzionamento per le diverse combinazioni di A, B, C.

Secondo metodo: uso delle tavole.

Tabella 4.2: Combinazioni possibili di tre termini nel binario

A	B	C	A	B*C	A+B*C	A+B	A+C	(A+B)*(A+C)
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0
0	1	0	0	0	0	1	0	0
0	1	1	0	1	1	1	1	1
1	0	0	1	0	1	1	1	1
1	0	1	1	0	1	1	1	1
1	1	0	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1

Essendo tre gli interruttori, abbiamo $2^3 = 8$ combinazioni di valori per A, B, C. I termini componenti la relazione sono 6 e cioè:

$$A, BC, A + BC, A + B, A + C, (A + B)(A + C)$$

Calcoliamo pertanto tutte le combinazioni possibili di A, B, C e degli altri 6 termini utilizzando le regole di definizione delle operazioni (+) e (*).

Le colonne corrispondenti ad A*B*C e (A+B)*(A+C) sono identiche e quindi le due espressioni in questione hanno gli stessi valori per tutte le combinazioni delle variabili A, B e C.

Da ciò discende l'equivalenza.

4.5 Esempi di applicazione

Esempio 1: Si consideri l'esempio in figura: esso è costituito dai tre relais A, B, C e 6 interruttori. È possibile trovare un circuito R equivalente nel senso che esso sarà chiuso quando R è chiuso, aperto quando R è aperto e che comporta un numero minimo di interruttori.

Il circuito R possiede tre rami in parallelo ognuno dei quali è composto di due interruttori in serie.

I rami hanno per funzioni di trasmissione:

$$A * B, \bar{A} * C, B * C$$

per cui la trasmissione T del circuito R sarà:

$$T = A * B + \bar{A} * C + B * C.$$

Sostituendo ad A , B, C, $\bar{A}$ i loro valori numerici, per ognuna delle $2^3 = 8$ combinazioni di A, B e C, si ottiene il valore della trasmissione in ogni caso.

Si possono eseguire le seguenti operazioni:

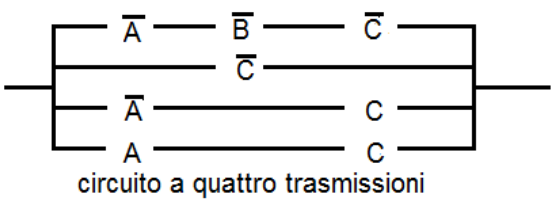
$$\begin{aligned} A * B + \bar{A} * C + B * C &= (A * B + \bar{A} * C) + B * C \\ &= (A * B + \bar{A} * C) + (\bar{A} + A) * (B * C) \\ &= (A * B + \bar{A} * C) + (\bar{A} * B * C + A * B * C) \\ &= (A * C + (\bar{A} * B * C + A * B * C)) + A * B \\ &= A * B + (\bar{A} * C(\bar{A} * C * B) + A * B * C \\ &= A * B + (\bar{A} * C + A * B * C) \\ &= A * B + (A * B * C + \bar{A} * C) \\ &= (A * B + A * B * C) + \bar{A} * C \\ &= A * B + \bar{A} * C \end{aligned} \tag{4.1}$$

Si ottiene quindi un nuovo circuito, equivalente al primo, ma costituito solo da 4 interruttori.

Volendo conoscere i valori della funzione T abbinata al circuito in questione, è sufficiente considerarla con l'ausilio di una tabella della verità:

A	B	C	$A * B$	$\bar{A} * C$	T_{R1}
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	0	1

Esempio 2: Si consideri il circuito in figura. Quando esso sarà aperto?



circuito a quattro trasmissioni

$$\begin{aligned} T &= \bar{A} * \bar{B} * \bar{C} + \bar{C} + \bar{A} * C + A * C \\ &= (\bar{A} * \bar{B} * \bar{C} + \bar{C}) + (\bar{A} * C + A * C) \\ &= \bar{C} + (\bar{A} * C + A * C) \\ &= \bar{C} + 1 * C \\ &= \bar{C} + C = 1 \\ T &= 1 \end{aligned} \tag{4.2}$$

Il circuito è dunque sempre chiuso ed è equivalente A una semplice connessione. Con questi esempi si è messo in evidenza come si possano sostituire i ragionamenti diretti sui circuiti con delle operazioni algebriche.

Negli esempi precedenti si è visto come, dato un circuito, mediante il disegno delle disposizioni dei suoi interruttori, se ne possa trovare l'espressione logica che lo rappresenta.

Su questa espressione logica si può operare utilizzando le proprietà R_1.... R_20 ottenendo così dei circuiti equivalenti: operando opportunamente si può pensare di ottenere circuiti con minimo numero di interruttori.

Si è visto inoltre come, volendo conoscere i valori assunti dalla funzione di trasmissione di un circuito, questi si possono dedurre utilizzando una tabella della verità.

Esiste ora il problema inverso: progettare un circuito che si comporti secondo una tabella della verità fissata A priori.

Tabella 4.3: Tabella della verità

A	B	C	T	
0	0	0	0	
0	0	1	1	T1
0	1	0	0	
0	1	1	1	T2
1	0	0	0	
1	0	1	0	
1	1	0	1	T3
1	1	1	1	T4

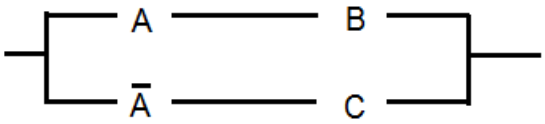
Sia allora noto il numero di interruttori necessari: tale informazione, e la tabella della verità, spiegano completamente il comportamento del circuito. Si consideri il circuito dato nell'esempio 1: di esso, in questo caso, si sa che èp fornito di tre interruttori A, B e C ed è data la sua tabella della verità.

Se si considera la funzione T, come somma delle funzioni T_1, T_2, T_3 e T_4, essa assumerà il valore 1 se, e soltanto se una delle funzioni T_1, T_2, T_3 e T_4, ha il valore 1. Così la funzione T varrà 1 sempre e soltanto in corrispondenza alle combinazioni prefissate. Se si considera ora una qualsiasi delle funzioni T_i (i=1...4) si nota che essa assumerà il valore 1 se la si rappresenta come prodotto degli interruttori: questi ultimi ovviamente saranno rappresentati nella forma non complementata se valgono 1, nella forma complementata se valgono 0. Nell'esempio precedente si ottiene:

$$T = \bar{A} * \bar{B} * C + \bar{A} * B * C + A * B * \bar{C} + A * B * C$$

Trovata l'espressione logica, si può cercare di semplificarla con i metodi studiati e, successivamente, disegnare il diagramma del circuito trovato.

$$\begin{aligned} T &= \bar{A} * \bar{B} * C + \bar{A} * B * C + A * B * + A * B(\bar{C} + C) \\ &= (\bar{A} * \bar{B} * C + \bar{A} * B * C) + A * B \\ &= \bar{A} * C(\bar{B} + B) + A * B \\ &= \bar{A} * C + A * B \end{aligned} \tag{4.3}$$



circuito costituito da 3 relais e 4 interruttori

Esempio 3: Con il seguente esempio si vuole illustrare il processo completo costituito dal prendere in considerazione un problema e trovarne le soluzioni in termini di dispositivi fisici. Un fattore ha un granaio la cui porta rimane facilmente aperta: la capra della fattoria ha la tendenza di avvicinarsi A questo deposito. Terzo elemento del problema è un lupo che circola nei pressi della fattoria. Il fattore decide allora di costruire un circuito con tre interruttori: uno che segnali se la porta del granaio è aperta o chiusa, uno per la **capra in vista**, uno per il **lupo in vista**.

Un garzone viene poi messo A controllare la situazione; se una o più delle circostanze predette si manifesta, egli deve premere il pulsante corrispondente. Se la situazione diventa pericolosa, il sistema darà l'allarme e il fattore si comporterà di conseguenza. È ovviamente il fattore, ideatore del circuito, che deve definire quando una situazione è pericolosa: egli potrà per esempio assumere che:

1. la situazione tranquilla se il lupo e la capra non sono contemporaneamente visibili.
2. il lupo non costituisce un pericolo per il grano.

Riassumendo queste considerazioni in una tabella della verità, si ottiene:

P	C	L	T
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

dove **L = lupo**, **C = capra**, **P = porta**.
Funzione di trasmissione:

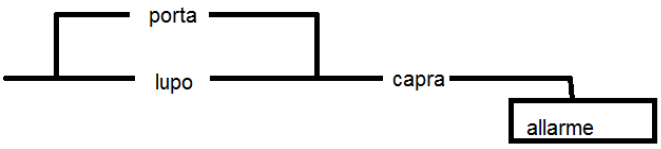
$$T = \bar{P} * L * C + P * C * \bar{L} + P * C * L$$

1. Disposizione 011- il lupo può mangiare la capra se non sono separati.
2. Disposizione 110- la porta è aperta e la capra è nelle vicinanze
3. Disposizione 111- entrambe le situazioni pericolose sono presenti.

Riducendo l'espressione della funzione di trasmissione si ottiene:

$$\bar{P} * C * L + P * C = C * (\bar{P} * L + P) = C * (P + L)$$

Il circuito cercato sarà quindi:



4.6 Funzioni di due variabili

Si sono definite in precedenza le due funzioni di due variabili, chiamate X*Y e X+Y, mediante l'uso di una tavola nella quale viene rappresentato il valore della funzione per ogni combinazione dei valori delle due variabili X e Y. Si può usare questo procedimento per definire altre funzioni di 2 variabili. per fare questo si consideri ancora la lista delle combinazioni di ingresso e si consideri il fatto che, per ognuna di esse, si può scegliere di attribuire come valore di uscita il valore 1 o il valore 0. Poiché esistono 4 combinazioni di ingresso, A ciascuna delle quali si può fare corrispondere 2 valori, in conseguenza esistono $2^4 = 16$ funzioni possibili di 2 variabili.

X	Y	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
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

Queste diverse funzioni possono ora essere espresse mediante le operazioni * e + e l'operazione di complementazione.

1) Si consideri per esempio la funzione T_1:

T_1 vale 1 quando X e Y valgono zero tutti e due, cioè quando $\bar{X}$ e $\bar{Y}$ valgono 1 e zero in tutti gli altri casi per cui:

$$T_1 = \bar{X} * \bar{Y}$$

Considerando la tavola della verità si può verificare che effettivamente la colonna corrispondente e T_1 e $\bar{X} * \bar{Y}$ hanno lo stesso valore per ogni combinazione dei valori di X e Y.

X	Y	T_1	$\bar{X}$	$\bar{Y}$	$\bar{X} * \bar{Y}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

In generale si otterrà:

$$
\begin{aligned}
T_0 &= 0 & T_8 &= X * Y \\
T_1 &= \bar{X} * \bar{Y} & T_9 &= X * Y + \bar{X} * \bar{Y} \\
T_2 &= \bar{X} * Y & T_{10} &= Y \\
T_3 &= \bar{X} & T_{11} &= \bar{X} + Y \\
T_4 &= X * \bar{Y} & T_{12} &= X \\
T_5 &= \bar{Y} & T_{13} &= X + \bar{Y} \\
T_6 &= X * \bar{Y} + \bar{X} * Y & T_{14} & \\
T_7 &= \bar{X} + \bar{Y} & T_{15} &= 1
\end{aligned}
$$

Si è così definita nell'insieme dei circuiti con interruttori un'algebra che viene chiamata **algebra dei contatti** o **algebra di commutazione**.

In seguito si introdurranno altri elementi di commutazione; principalmente elementi elettronici e magnetici che si possono organizzare in reti complesse e che potranno essere4 studiati grazie ad un'algebra che si rivelerà avere la stessa struttura dell'algebra dei contatti.

5

Algebra delle classi - Algebra della logica

5.1 Insiemi

Introduciamo il concetto di insieme mediante esempi.

L'insieme degli italiani, l'insieme dei numeri interi, ecc. Gli enti che fanno parte di un insieme vengono chiamati **elementi**.

Per scrivere che un elemento X appartiene ad un **insieme A**, si usa la notazione:

$$X \in A$$

Inversamente, se X non appartiene ad A si scriverà:

$$X \notin A$$

Inoltre, per rappresentare un insieme, si utilizzeranno delle parentesi, all'interno delle quali si troveranno gli elementi stessi, oppure il simbolo dell'elemento generale con l'enuncuato della proprietà comune A tutti gli elementi dell'insieme.

Così se R è l'insieme delle radici della equazione.

$$x^3 - 6x^2 + 11x - 6 = 0$$

si avrà:

$$R = 1, 2, 3$$

oppure:

$$R = \{x | x^3 - 6x^2 + 11x - 6 = 0\}$$

e questa seconda notazione si leggerà: R è l'insieme degli elementi X tali che si abbia:

$$x^3 - 6x^2 + 11x - 6 = 0$$

Si dice che due insiemi, A e B, sono uguali (0 identici) se essi sono formati dagli stessi elementi: e si scrive $A = B$.

In altri termini, ogni elemento X appartenente ad A appartiene A B e viceversa.

Esempio 1. Gli insiemi:

$$\begin{cases} A = \{x \mid e\ un\ numero\ pari\} \\ B = 2 \end{cases}$$

dove x è l'insieme costituito dal solo elemento 2, sono uguali.

Si dice che un insieme A è un sottoinsieme di un insieme Bse ogni elemento di A appartiene A B (si dice anche che A è incluso in B o che A è una parte di B, o infine, che B contiene A) e si utilizzano le notazioni:

$$1) \qquad A \subseteq B$$

$$2) \qquad B \supseteq A$$

Nella definizione data, come nelle notazioni 1) e 2), non si esclude il caso in Dato un insieme A come sottoinsieme di un insieme U, si chiama **complemento di** A **rispetto ad U** l'insieme dei punti di U che non appartengono ad A.

Si scrive:

$$U - A \qquad oppure \qquad C_u(A)$$

Si può scrivere, utilizzando le notazioni di cui sopra:

$$U - A = \{x |\ x \in U \qquad e \qquad x \notin A\}$$

Esempio 2. Se U è l'insieme dei numeri reali e A quello dei numeri razionali, U-A è l'insieme dei numeri irrazionali.

L'insieme U-A è un complemento relativo. Quando non vi è ambiguità sull'insieme U da considerare, si può definire un **complemento assoluto** come l'insieme dei punti che non appartengono ad A e lo si denota con la espressione:

$$\bar{A} = \{x \mid x\ \notin A\}$$

Si noti che l'operazione con la quale si ottiene $\bar{A}$ da A, cioè la complementazione, è una operazione ad un solo termine.

Definiamo ora operazioni A 2 termini (binarie).

a) L'unione di due termini A e B è l'insieme dei punti che appartengono ad A o A B oppure ad entrambi contemporaneamente.

L'unione di A e B è rappresentata:

$$A \cup B = \{x \mid x \in A \quad oppure \quad x \in B \quad oppure \quad x \in A \quad e \quad x \in B\}$$

Così l'unione dell'insieme dei punti del segmento $0 \leq x \leq 1$ e del segmento $0,5 \leq x \leq 1,5$ è l'insieme dei punti del segmento $0 \leq x \leq 1,5$.

b) L'intersezione di due insiemi A e B è l'insieme dei punti che appartengono contemporaneamente ad A ed A B e si rappresenta:

$$A \cap B = \{x \mid x \in A \quad e \quad x \in B\}$$

Per esempio l'intersezione dei numeri primi e dell'insieme dei numeri pari è l'insieme $\{2\}$.

È comodo introdurre un insieme il quale non possiede alcun elemento: esso si chiama **elemento vuoto** e lo si rappresenta con il simbolo Φ.

Segue che l'uguaglianza

$$A \cap B = \Phi$$

significa che gli insiemi A e B non hanno alcun punto in comune. Si dice anche che essi sono disgiunti. Analogamente è conveniente introdurre un insieme contenente tutti gli elementi possibili: insieme universale o universo (e lo si noterà con il simbolo $U.I$, ecc.).

Dato un insieme E, i suoi sottoinsiemi possono essere considerati come gli elementi dell'insieme potenza $P(E)$ o 2^E (insieme delle parti di E costituito dai sottoinsiemi di E, inclusi l'insieme vuoto e quello improprio).

Esempio: $E = \{1, 2, 3\}$

$$P(E) = \{\Phi, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

P(E) è dunque un insieme di insiemi dei suoi sottoinsiemi: sono particolarmente interessanti quelli chiamati **ricoprenti di** E o parti di E.

Dato un insieme E (appartenente all'insieme universale U), si chiama ricoprimento di E un insieme delle parti di E un insieme delle parti di E tale che E sia uguale alla loro unione.

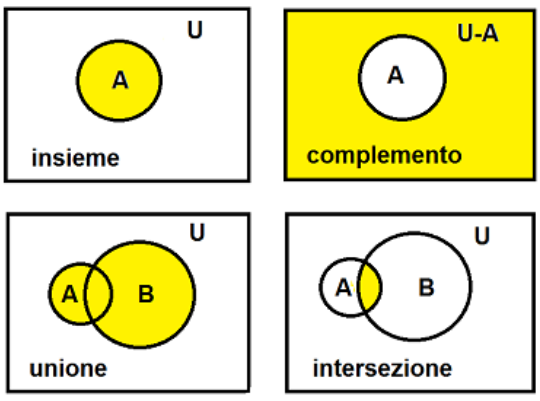
Esempio. L'insieme dei numeri interi, dei numeri razionali, dei numeri irrazionali e dei numeri trascendenti, costituisce un ricoprimento dell'insieme dei numeri reali.

Un ricoprimento di E sia tale che due insiemi qualunque di esso siano disgiunti. Ogni punto di E si trova allora in un insieme della partizione, e uno solo.

Così l'insieme dei numeri razionali e dei numeri irrazionali, costituiscono una partizione dell'insieme dei reali.

Queste differenti definizioni possono ricevere una interpretazione grafica chiamata **diagrammi di Venn**. L'insieme universale è rappresentato con un rettangolo e gli insiemi A, B ecc. sono rappresentati come **cerchi**.

Nelle figure seguenti si dà l'interpretazione delle diverse operazioni definite precedentemente:



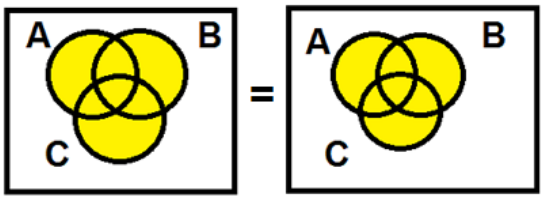
5.2 Algebra delle classi

Si consideri un insieme I che starà A rappresentare, nel seguito, l'insieme universale. I differenti sottoinsiemi di I formano essi stessi un insieme, sul quale si possono definire le tre operazioni di: **unione**, **intersezione** e **complementazione**. L'insieme dei sottoinsiemi di I costituisce così un'algebra denominata **algebra delle classi**.

A partire dalle definizioni di insieme, sottoinsieme e delle 3 operazioni di cui sopra, si possono dimostrare diverse proprietà di questa algebra. Esse si possono giustificare con l'aiuto dei **diagrammi di Venn**, oppure con rigorose dimostrazioni, partendo dalle definizioni.

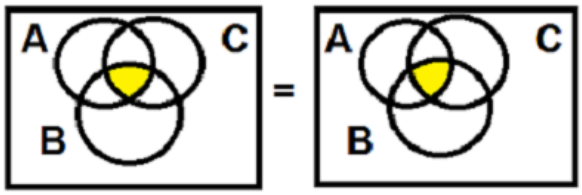
$$E_1) \qquad (A \cup B) \cup C = A \cup (B \cup C)$$

È la proprietà detta dell'associatività dell'unione $\cup$.



$$E_2) \qquad (A \cap B) \cap C = A \cap (B \cap C)$$

È la legge d'associatività dell'operazione di intersezione.



$$E_3) \qquad A \cup B = B \cup A$$

Essa è la legge di **commutatività** dell'operazione di unione $\cup$; risulta dalla definizione di A$\cup$B: il risultato non dipende evidentemente dall'ordine dei due operandi A e B.

$$E_4) \qquad A \cap B = B \cap A$$

Legge di commutatività dell'operazione d'intersezione.

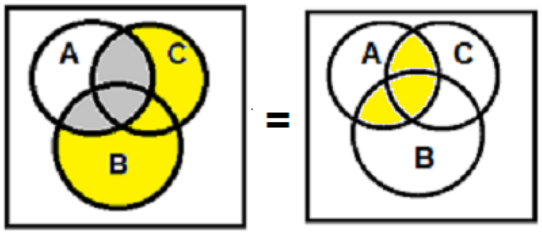
$$E_5) \qquad A \cup \Phi = A$$

Φ non contiene nessun elemento quindi la sua unione con l'insieme A non è altro che A stesso.

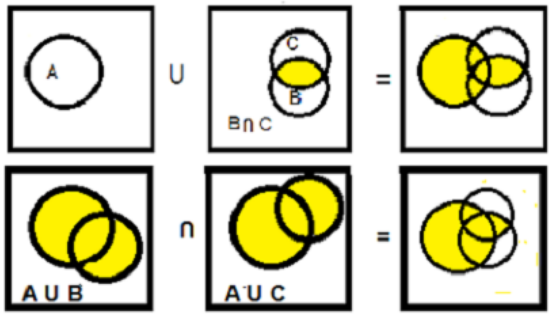
$$E_6) \qquad A \cap I = A$$

Essendo A contenuto in I (per definizione), i punti comuni ad A e I sono tutti quelli di A.

$$E_7) \qquad A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$



$$E_8) \qquad A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$



Legge di distributività dell'unione rispetto all'intersezione.

$$E_9) \qquad A \cap \bar{A} = \Phi$$

Un insieme A ed il suo complementare $\bar{A}$ non hanno alcun punto in comune : essa è una conseguenza della definizione di $\bar{A}$

$$E_{10}) \qquad A \cup \bar{A} = I$$

L'unione di un insieme A e del suo complementare $\bar{A}$ è sempre l'insieme universale. Esso è una conseguenza della definizione di $\bar{A}$. quindi il complementare $\bar{A}$ di A soddisfa alle due relazioni : E_9 e E_{10}.

Inversamente, se un insieme B soddisfa alle relazioni:

$$E_9 \quad A \cap B = \Phi \quad e \quad E_{10} \quad A \cup B = I$$

Esso è il complemento di A; ossia:

$$B = \bar{A}$$

Infatti, sia un elemento $x \in B$, la E'_9 afferma che $x \notin A$ e la $E'_1 0$ che $x \in I$.

Dunque X appartiene ad $I - A = \bar{A}$, $x \in \bar{A}$. Dunque $B \subset \bar{A}$. Ma ogni elemento di A, d'altra parte, verifica E_9 ed $E_1 0$; dunque se $x \in \bar{A}$, $x \in B$; per cui $\bar{A} \subset B$.

Da cui $\bar{A} = B$. In altri termini, le due relazioni E'_9 e $E'_1 0$ hanno per soluzioni $\bar{A}$ e le si utilizzeranno, in pratica, per definire direttamente $\bar{A}$.

$$E_{11}) \qquad A \cup A = A$$

Legge di idempotenza per la unione U.

$$E_{12}) \qquad A \cap A = A$$

Legge di idempotenza per la intersezione ∩.

$$E_{13}) \qquad A \cup I = I$$
$$E_{14}) \qquad A \cap \Phi = \Phi$$
$$E_{15}) \qquad A \cup (A \cap B) = A$$

Prima legge d'assorbimento.

L'intersezione E =A∩B è un sottoinsieme di A; l'ulteriore unione di E con A non può che dare A.

$$E_{16}) \qquad A \cap (A \cup B) = A$$

Seconda legge di assorbimennto.

A è contenuto in $E = A \cup B$, da cui $A \cap E = A$.

$$E_{17}) \qquad A \cup (\bar{A} \cap B) = A \cup B$$
$$E_{18}) \qquad A \cap A(\bar{A} \cup B) = A \cap B$$
$$E_{19}) \qquad \overline{A \cup B} = \bar{A} \cap \bar{B}$$

Primo teorema di Morgan per gli insiemi: esso si enuncia dicendo che il complementare della unione è uguale all'intersezione dei complementari.

Secondo teorema di De Morgan:

$$E_{21}) \qquad \overline{A \cup B} = \bar{A} \cap \bar{B}$$

Il complemento delle intersezioni è uguale all'unione dei complementari.

5.3 Algebra della logica

5.3.1 Proposizioni: valore logico delle proposizioni

Se si esamina il linguaggio di cui si fa uso ogni giorno, e più particolarmente quello che serve come mezzo per esprimere un ragionamento scientifico, se ne possono discernere degli elementi detti **proposizioni**: legandole tra di loro, si può costruire l'enunciato di un teorema, la dimostrazione di un teorema, una definizione, ecc..

Questa articolazione del linguaggio in proposizioni è, del resto, una delle caratteristiche fondamentali del linguaggio umano.

Così le espressioni:

1. "11 è un numero primo"
2. "il triangolo ABC è rettangolo"

costituiscono delle **proposizioni**.

Ciascuna di queste proposizioni, può essere esaminata sotto l'aspetto della sua verità o della sua falsità.

Per esempio: la proposizione 1) è vera mentre la 2) può essere esaminata con il criterio su accennato e deve essere possibile se essa è vera o falsa.

Nel seguito si studieranno le proposizioni e si supporrà che le proposizioni esaminate non possano essere che vere o false.

5.3.2 Operazioni sulle proposizioni

Continuando l'esame del linguaggio ordinario, si può mettere in evidenza tutta una classe di parole, o anche di espressioni più complesse, che però non sono proposizioni ma elementi di legame tra proposizioni: **e**, **o**, **ma**, **oppure**, **"implica che"**, **"equivale a dire che"**, ecc. Nel linguaggio corrente, queste operazioni di legame delle proposizioni elementari, formando proposizioni più complesse, non sono prive di ambiguità.

Nondimeno, le nozioni di proposizione e di legami, forniscono lo spunto per un insieme di concetti matematici che, in questo caso, saranno dotati di una definizione rigorosa.

5.3.3 Prodotto logico: operazione AND

Consideriamo per esempio le due proposizioni:

- "x è un numero primo"
- "Il triangolo ABC è rettangolo"

Con l'aiuto della congiunzione **"e"** (AND) si può formare una nuova proposizione che si enuncerà: **" X è un numero primo E il triangolo ABC è rettangolo"**.

Quando essa sarà vera?

Questa nuova proposizione sarà considerata come vera, quando le due proposizioni componenti sono vere tutte e due, e falsa in tutti gli altri casi. Ovviamente si sarebbe potuto considerare altre due proposizioni, ad esempio:

- "Roma è la capitale d'Italia"
- "Il clima italiano è mite"

e costruire in maniera analoga una proposizione nuova con la congiunzione **"e"** (AND).

Si vede dunque che l'operazione **AND** da una proposizione nuova, A partire da due proposizioni componenti, in una maniera caratterizzata essenzialmente dal modo in cui si determina se essa è falsa o vera, conoscendo la falsità o la verità delle componenti, ma, in fondo, indipendentemente dal senso e dalla natura delle proposizioni. Si può aòòora rappresentare ogni proposizione, con una variabile capace di prendere soltanto i due valori logici rappresentati von **V** (vero) e **F** (falso).

L'operazione di AND allora, tra due proposizioni generiche P e Q verrà definita dalla tavola AND.

Si definisce così una delle operazioni della logica matematica: essa viene indicata con il simbolo $\wedge$ e si chiama operazione di AND o prodotto logico. Una tavola come quella A fianco prende il nome di **tavola della Verità**: essa descrive, per tutte le combinazioni possibili, il valore (vero o falso) della proposizione composta, in funzone dei valori delle proposizioni componenti.

5.3.4 Operazione di OR inclusivo (somma logica)

Tabella 5.1: Tavolta della verità OR

P	Q	$P \vee Q$
F	F	F
F	V	V
V	F	V
V	V	V

Si consideri la frase (nell'algebra ordinaria)

> "il prodotto XY è nullo se $x = 0$, $y \neq 0$; oppure se $x \neq 0$, $y = 0$; oppure $x = 0$', $y = 0$."

La proposizione $(x = 0)$ oppure $(y = 0)$ fornisce un esempio di una nuova operazione che si può definire rigorosamente mediante una tavola

Date due proposizioni P e Q si chiamerà **"proposizione P OR Q"** e si scriverà $P \vee Q$, la proposizione composta definita dalla tavola affiancata.

L'operazione $\vee$, così definita, prende il nome di **somma logica**.

5.3.5 Negazione di una proposizione

P	$\neg P$
F	V
V	F

A partire da una proposizione come **"oggi piove"**, possiamo formulare una seconda proposizione **"oggi non piove"**. È chiaro che se la prima è vera, la seconda è falsa e viceversa. Si può quindi definire l'operazione di negazione: la negazione di una proposizione P viene rappresentata con il simbolo $()P$; ed è data dalla tavola affiancata. Conformemente all'uso generale, si preferisce rappresentare i due valori logici di una variabile, non con i simboli **V** ed **F** ma con i simboli 0 ed 1; si noti però che si tratta di una convenzione pratica alla quale l'aritmetica è completamente estranea. Tra le due convenzioni possibili si è scelta la seguente:

$$V = 1 \qquad F = 0$$

È facile riscrivere la tavole precedenti con il nuovo simbolismo.

Oltre alle tre operazioni logiche fin qui studiate, possiamo, con lo stesso metodo di definizione, costruire $2^4 = 16$ tavole della verità di due variabili che definiscono altrettante operazioni o funzioni logiche:

Tabella 5.2: Funzioni logiche

		F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
P	Q	$P \downarrow Q$		non P		$\sqrt{P}$	$P \oplus Q$	$P\|Q$	$P \wedge Q$	$P \leftrightarrow Q$	Q	$P \rightarrow Q$	P	$Q \rightarrow P$	$P \vee Q$		
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

5.3.6 Funzione OR esclusiva (F6)

È il nome dato alla funzione **F6**: si confrontino le due tavole della verità di $P \oplus Q$ e $P \vee Q$

Si noti che $P \oplus Q$ ha il valore 1 quando P e Q hanno valori logici differenti e 0 altrimenti.

Al contrario la proposizione $P \vee Q$ assume il valore 1 anche per $P = 1$ e $Q = 1$

Tabella 5.3: Tavola della verità OR

		F6	F14
P	Q	$P \oplus Q$	$P \vee Q$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1

Esempio. Il prodotto delle variabili algebriche non nulle A e B è negativo se A oppure B sono negative.

5.3.7 Equivalenza $P \leftrightarrow Q$ (F9)

Con l'operazione $\leftrightarrow$ si otterrà una nuova proposizione e si scriverà $P \leftrightarrow Q$; vera quando P e Q hanno lo stesso valore logico e falsa nel caso contrario: essa si chiama equivalenza di P e Q; oppure implicazione reciproca di P e Q.

Si noti come le due proposizioni $P \leftrightarrow Q$ e $P \oplus Q$ sono l'una la negazione dell'altra. Il nome di implicazione di $P \leftrightarrow Q$ trova giustificazione nella operazione seguente.

Tabella 5.4: Equivalenza (F9)

		F9	F6
P	Q	$P \leftrightarrow Q$	$P \oplus Q$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0

5.3.8 Implicazione (F11)

È la funzione $P \rightarrow Q$. Essa traduce il senso, talvolta vago, della nozione di implicazione nel linguaggio ordinario: quando P è vero, anche Q è vero (se piove allora ci sono le nubi): quindi $P \rightarrow Q$ è vero se $P = 1$ e $Q = 1$.

P	Q	$P \rightarrow Q$
0	0	1
0	1	1
1	0	0
1	1	1

Inoltre, quando P è falso, niente impedisce che Q assuma un valore arbitrario 1 o 0 e quindi $P \to Q$ è considerata vera nei due casi in cui $P = 0$. In breve: Q non può essere falsa quando P è vera.

Il simbolo $P \to Q$ si legge: **P implica Q** oppure **P è una condizione sufficiente per avere Q**.

		F11	F13	F9
P	Q	$P \to Q$	$Q \to P$	$(P \to Q) \wedge (Q \to P)$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	1	1	1

La funzione **F13** rappresenta in maniera analoga la funzione $Q \to P$. Si consideri la proposizione $(P \to Q) \wedge (Q \to P)$: si può costruire la tabella della verità affiancata.

La colonna di destra è la stessa di quella della operazaione $P \leftrightarrow Q$ ciò che giustifica il simbolo e il nome di implicazione reciproca della operazione $\leftrightarrow$.

5.3.9 Funzione di Scheffer P/Q o incompatibilità (NAND)

Tabella 5.5: F7 - incompatibilità logica

		F7	F8
P	Q	P/Q	$P \wedge Q$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

È la funzione **F7**. La proposizione P/Q è falsa quando P e Q sono entrambi veri, e vera in tutti gli altri casi. Questo giustifica il nome di **incompatibilità** di P e Q. Essa è anche uguale alla negazione della funzione **F8**.

5.3.10 Funzione di Peirce (funzione NOR) $P \downarrow Q$

		F1	F14
P	Q	P_Q	$P \vee Q$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

È la funzione **F1**. La proposizione $P \downarrow Q$ è vera quando né P né Q sono vere. Essa è anche uguale a:

$$\overline{P \vee Q}$$

5.3.11 Tautologie e contraddizioni

Una *proposizione composta* che è falsa per tutti i valori logici possibili delle n proposizioni componenti, è una contraddizione.

Una *proposizione composta* che è vera per tutti i valori logici delle n proposizioni componenti, è una tautologia.

Esempio 1. *Principio della doppia negazione (tautologia)*: ogni proposizione equivale alla negazione della sua negazione.

$$P \leftrightarrow \bar{\bar{P}}$$

P	$\bar{P}$	$\bar{\bar{P}}$	$P \leftrightarrow \bar{\bar{P}}$
0	1	0	1
1	0	1	1

Esempio 2. *Principio d'identità*: ogni proposizione implica se stessa.

P	P	$P \leftrightarrow P$
0	0	1
1	1	1

Esempio 3. *Principio del terzo escluso*: ogni proposizione è **vera** o **falsa**.

$$P \vee \bar{P}$$

P	$\bar{P}$	$P \vee \bar{P}$
0	1	1
1	0	1

Esempio 4. $P \wedge \bar{P}$: contraddizione.

- Ogni proposizione non può essere contemporaneamente vera o falsa.
- Qualche proprietà delle funzioni AND, OR e **negazione**.
- Nel seduito compaiono un certo numero di identità algebriche che legano le tre operazioni AND, OR e **negazione** ($\wedge$, $\vee$, $-$).

Esse si presentano sotto forma di tautologie e di contraddizioni che figurano in certe identià.

L_1	$(P \vee Q) \vee R \leftrightarrow P \vee (Q \vee R)$	associativita dell'operazione OR
L_2	$(P \wedge Q) \wedge R \leftrightarrow P \wedge (Q \wedge R)$	associativita dell'operazione AND
L_3	$P \vee Q \leftrightarrow Q \vee P$	commutativita dell'operazione OR

L_4	$P \wedge Q \leftrightarrow Q \wedge P$	commutativita dell'operazione AND
L_5	$P \vee P$	tautologia
L_6	$P \wedge P$	contraddizione
L_7	$P \wedge (Q \vee R) \leftrightarrow (P \wedge Q) \vee (P \wedge R)$	distributiva di AND rispetto ad OR
L_8	$P \vee (Q \wedge R) \leftrightarrow (P \vee Q) \wedge (P \vee R)$	distributiva di OR rispetto ad AND
L_9	$\neg(P \wedge \neg P)$	principio di contraddizione
L_{10}	$P \vee \neg P$	principio del terzo escluso
L_{11}	$P \vee P \leftrightarrow P$	idempotenza di OR
L_{15}	$P \vee (P \wedge Q) \leftrightarrow P$	assorbimento
L_{16}	$P \wedge (P \vee Q) \leftrightarrow P$	assorbimento
L_{17}	$P \vee (\neg P \wedge Q) \leftrightarrow P \vee Q$	
L_{18}	$P \wedge (\neg P \vee Q) \leftrightarrow P \wedge Q$	
L_{19}	$\neg(P \vee Q) \leftrightarrow (\neg P \wedge \neg Q)$	teorema di Morgan
L_{20}	$\neg(P \wedge Q) \leftrightarrow (\neg P \vee \neg Q)$	teorema di Morgan

5.3.12 Esempi d'applicazione

Nullità di un prodotto algebrico ordinario

Nell'algebra ordinaria perché un prodotto $m = a \times b \times c$ sia $\neq 0$ è necessario e sufficiente che sia $a \neq 0$, $b \neq 0$, $c \neq 0$.

Si ha dunque:

$$(a \times b \times c \neq 0) \leftrightarrow (a \neq 0) \wedge (b \neq 0) \wedge (c \neq 0)$$

Inversamente perché sia $m = 0$:

$$(a \times b \times c = 0) \leftrightarrow (a = 0) \vee (b = 0) \vee (c = 0)$$

Regole dei segni nell'algebra ordinaria

Si consideri il prodotto $M =$A$\times b$

$$a \neq 0 \qquad b \neq 0$$

La regola dei segni si scriverà:

$$\begin{gathered}(a \times b > 0) \leftrightarrow [(a < 0) \wedge (b < 0)] \vee [(a > 0) \wedge (b > 0)] \\ (a \times b > 0) \leftrightarrow [(a < 0) \leftrightarrow (b < 0)] \\ (a \times b > 0) \leftrightarrow [(a > 0) \leftrightarrow (b < 0)]\end{gathered} \tag{5.1}$$

Inversamente:

$$\begin{gathered}(a \times b < 0) \leftrightarrow [(a < 0) \oplus (b < 0)] \\ (a \times b < 0) \leftrightarrow [(a > 0) \oplus (b > 0)] \\ (a \times b < 0) \leftrightarrow [(a < 0) \leftrightarrow (b > 0)]\end{gathered} \tag{5.2}$$

6

Algebre booleane

6.1 Definizioni assiomatiche

Si chiamera algebra di Boole o algebra booleana il sistema:

$$B =< E, (+), (\cdot) \ , 0, 1, >$$

dove:

1. E è un insieme non vuoto che contiene almeno due elementi, e cioè 1 e 0,
2. $(+)$ e $(\cdot)$ sono due operazioni binarie definite su E e tali che E sia chiuso rispetto ad esse.
3. su E è definita una relazione di equivalenza, rappresentata dalla notazione $=$ (e si legge uguale), e come tale soddisfacente, $\forall x, y, z \in E$ alle seguenti relazioni:
   a) $(x = y) \leftrightarrow (y = x)$ simmetria
   b) $X = X$ riflessivata
   c) $(x = y) \wedge (y = z) \rightarrow (x = z)$ transitivita

e tale che, $\forall\ A,\ B,\ C,\ \in E$, risultano soddisfatti i seguenti assiomi:

Assiomi $A_{i,j}$			
	associativita		
A1.1	$(A + B) + C = A + (B + C)$	A1.2	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$
	commutativita		
A2.1	$A + B = B + A$	A2.2	$A \cdot B = B \cdot A$
	distributivita		
A3.1	$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	A3.2	$A \cdot B = B \cdot A$
A4.1	$A + 0 = 0 + A$	A4.2	$A \cdot 1 = 1 \cdot A = A$

Nei precedenti due capitoli, abbiamo studiato l'algebra dei circuiti, l'algebra delle classi e l'algebra della logica; tutte e tre queste algebre rappresentano esempi di strutture algebriche Booleane. La definizione assiomatica dell'algebra di Boole riassume infatti le regole di calcolo incontrate nello studio delle algebre su menzionate. Per quanto riguarda l'applicazione dell'algebra astratta di Boole, vogliamo fare un breve cenno ad alcuni risultati dell'algebra dei circuiti (più avanti considereremo esempi di applicazione). Partendo dalla constatazione che operazioni quali "connettere in serie", "connettere in parallelo" due o più interruttori elettrici e "invertire"

un interruttore (chiuderlo se esso è aperto, aprirlo se esso è chiuso), soddisfano tutti gli assiomi $A_i, j(i = 1, .., 5); J = 1, 2)$ si giunge alla conclusione che ogni risultato, ottenuto in sede astratta per le algebre di Boole, può essere tradotto in termini di interruttori, o più in generale, di circuiti.

6.2 Relazioni fondamentali in una algebra di Boole

Gli assiomi precedenti ci permettono di dimostrare diversi teoremi. Per semplificare le dimostrazioni si scriverà a destra di ogni identità o equazione, gli assiomi o i teoremi utilizzati.

Si hanno i seguenti teoremi:

$\underline{T1}$	$A + A = A$	infatti
	$A + A = (A + A) \cdot 1$	$(A4.2)$
	$= (A + A) \cdot (A + \bar{A})$	$(A5.1)$
	$= A + (A \cdot \bar{A})$	$(A3.2)$
	$= A + 0$	$(A5.2)$
	$= A$	$(A4.1)$

In maniera più generale si può dimostrare (per ricorrenza per esempio) che:

$$A + A + A..... + A = A$$

$\underline{T2}$	$A \cdot A = A$	infatti
	$A \cdot A = (A \cdot A) + 0$	$(A4.1)$
	$A \cdot A = (A \cdot A) + (A \cdot \bar{A})$	$(A5.2)$
	$= A \cdot (A + \bar{A})$	$(A3.1)$
	$= A \cdot 1$	$(A5.1)$
	$= A$	$(A4.2)$

In maniera analoga si potrà generalizzare il risultato nella maniera seguente:

$$A \cdot A \cdot A..... \cdot A = A$$

(T1) e (T2) costituiscono le proprietà di idem potenza; ossia che in un'algebra di Boole non vi sono ne multipli ne potenze.

$\underline{T3}$	$A + 1 = 1$	infatti
	$A + 1 = (A + 1) \cdot 1$	$(A4.2)$
	$= (A + 1) \cdot (A + \bar{A})$	$(A5.1)$
	$= A + (1 \cdot \bar{A})$	$(A3.2)$
	$= A + \bar{A}$	$(A4.2)$
	$= 1$	$(A5.1)$

$\underline{T4}$	$\underline{A \cdot 0 = 0}$	infatti
	$A \cdot 0 = (A \cdot 0) + 0$	$(A4.1)$
	$= (A \cdot 0) + (A \cdot \bar{A})$	$(A5.2)$
	$= A \cdot (0 + \bar{A})$	$(A3.1)$
	$= A \cdot \bar{A})$	$(A4.1)$
	$= 0$	$(A5.2)$

$\underline{T5}$	$\underline{A + (A \cdot B) = A}$	(legge di assorbimento)	infatti
	$A + (A \cdot B) = (A \cdot 1) + (A \cdot B)$		$(A4.2)$
	$A \cdot (1 + B)$		$(A3.1)$
	$A \cdot 1$		$(T3)$
	A		$(A4.2)$

$\underline{T6}$	$\underline{A \cdot (A + B) = A}$	(legge di assorbimento)	infatti
	$A \cdot (A + B) = (A \cdot A) + (A \cdot B)$		(3.1)
	$A + (A \cdot B)$		$(T2)$
	A		$(T5)$

$\underline{T7}$	$\underline{A + (\bar{A} \cdot B) = A + B}$	infatti
	$A + (\bar{A} \cdot B) = (A + \bar{A}) \cdot (A + B)$	$(A3.2)$
	$1 \cdot (A + B)$	$(A5.1)$
	$A + B$	$(A4.2)$

$\underline{T8}$	$\underline{A \cdot (\bar{A} + B) = A \cdot B}$	infatti
	$A \cdot (\bar{A} + B)) = (A \cdot \bar{A}) + (A \cdot B)$	$(A3.1)$
	$0 + (A \cdot B)$	$(A5.2)$
	$A \cdot B$	$(A4.1)$

$\underline{T9}$ Il complemento $\bar{A}$ di un elemento A è unico $\quad \bar{A}_1 = \bar{A}_1 \cdot 1 \quad (A4.2)$

Ragionando per assurdo, si supponga che l'elemento a possieda due componenti $\bar{A}_1$ e $\bar{A}_2$. In virtù dell'assioma (A5) si avranno le relazioni:

$$\begin{cases} A + \bar{A}_1 = 1 \\ A \cdot \bar{A}_1 = 0 \end{cases}$$

$$\begin{cases} A + \bar{A}_2 = 1 \\ A \cdot \bar{A}_2 = 0 \end{cases}$$

Si ha allora:

$$\begin{aligned} \bar{A}_1 &= \bar{A}_1 \cdot 1 && (A4.2) \\ &= \bar{A}_1 \cdot (A + \bar{A}_2) && (A5.1) \\ &= \bar{A}_1 \cdot A + \bar{A}_1 \cdot \bar{A}_2 && (A3.1) \\ &= 0 + \bar{A}_1 \cdot \bar{A}_2 && (A5.2) \\ &= A \cdot \bar{A}_2 + \bar{A}_1 \cdot \bar{A}_2 && (A5.2) \\ &= \bar{A}_2 \cdot A + \bar{A}_2 \cdot \bar{A}_1 && (A2.2) \\ &= \bar{A}_2 \cdot (A + \bar{A}_1) && (A3.1) \\ &= \bar{A}_2 \cdot 1 && (A5.1) \\ &= \bar{A}_2 && (A4.2) \end{aligned}$$

Quindi A_1 e A_2 non possono che essere uguali.

Si chiama **complementazione** l'operazione che consiste nel far corrispondere l'elemento $\bar{A}$ ad A.

$$\underline{T10} \quad \bar{\bar{A}} = A \qquad\qquad \textit{involuzione}$$

Si consideri l'elemento $\bar{A}$ e si cerchi il suo complemento che verrà scritto $\bar{\bar{A}}$.

Si avrà:

$$\begin{cases} A + \bar{\bar{A}} = 1 \\ \bar{A} \cdot \bar{\bar{A}} = 0 \end{cases}$$

d'altra parte:

$$\begin{cases} A + \bar{A} = 1 \\ A \cdot \bar{A} = 0 \end{cases}$$

Si ha allora che $\bar{A}$ e $\bar{\bar{A}}$ soddisfano alle stesse equazioni e poiché per il teorema T9, il complemento è unico, segue che $\bar{\bar{A}} = A$

$$\underline{T11} \quad \bar{(A + B)} = \bar{A} \cdot \bar{B} \qquad\qquad \text{primo teorema di de Morgan}$$

Si cercherà il complemento di$\bar{A} \cdot \bar{B}$ e si dimostrerà che esso è $A + B$.

Si consideri la quantità $(A + B) + (\bar{A} \cdot \bar{B})$:

$$\begin{aligned} (A + B) + (\bar{A} \cdot \bar{B}) &= [(A + B) + \bar{A}] \cdot [(A + B) + \bar{B}] \\ &= [A + (B + \bar{A})] \cdot [A + (B + \bar{B})] \\ &= [(A + \bar{A}) + B] \cdot [A + (B + \bar{B})] \\ &= (1 + B) \cdot (A + 1) \\ &= 1 \cdot 1 \end{aligned}$$

Analogamente si ha:

$$\begin{aligned}(A+B)\cdot(\bar{A}\cdot\bar{B}) &= (\bar{A}\cdot\bar{B})\cdot(A+B)\\ &= (\bar{A}\cdot\bar{B})\cdot A + (\bar{A}\cdot\bar{B})\cdot B\\ &= (\bar{A}\cdot A)\cdot\bar{B} + \bar{A}\cdot(\bar{B}\cdot B)\\ &= 0\cdot\bar{B} + \bar{A}\cdot 0\\ &= 0\end{aligned}$$

Dunque $x = A + B$ e $y = \bar{A}\cdot\bar{B}$ verificano le due equazioni:

$$\begin{cases} x + y = 1 \\ x \cdot y = 0 \end{cases}$$

per cui $A + B$ è il complemento (unico) di $\bar{A}\cdot\bar{B}$: dunque

$$\begin{aligned}A + B &= \overline{(\bar{A}\cdot\bar{B})}\\ \overline{(a+B)} &= \overline{\overline{(\bar{A}\cdot\bar{B})}} = \bar{A}\cdot\bar{B}\\ \overline{(A+B)} &= \bar{A}\cdot\bar{B}\end{aligned}$$

$\underline{T12}$ $\bar{}(A+B) = \bar{A}\cdot\bar{B}$ secondo teorema di de Morgan

Si consideri il secondo membro e se ne prenda il complemento:

$$\begin{aligned}\overline{(\bar{A}+\bar{B})} &= \bar{\bar{A}}\cdot\bar{\bar{B}} && (T11)\\ &= A\cdot B && (T10)\end{aligned}$$

Si consideri ancora il complemento dei due membri:

$$\begin{aligned}\overline{\overline{(\bar{A}+\bar{B})}} &= \overline{A\cdot B}\\ \bar{A}+\bar{B} &= \overline{A\cdot B} && (T10)\end{aligned}$$

6.3 Espressioni duali - Principio di dualità

Si considerino le coppie degli assiomi $Ai.j$. Due assiomi di una qualunque di queste coppie si corrispondono nella maniera indicata nella tabella seguente (dove X rappresenta la variabile generica che figura negli assiomi).

Tabella 6.1: Regole di dualizzazione

$+$	$\cdot$
$\cdot$	$+$
0	1
1	0
x	x
x	x

Con questo procedimento, da un assioma di una coppia, si deduce l'altro.
Per esempio, a partire da A1.1:

$$A + (B + C) = (A + B) + C$$

si può scrivere immediatamente A1.2

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

Si dice che i due assiomi di una tale coppia sono duali: oppure che le identità che li esprimono sono duali. In generale, si chiameranno **espressioni duali** due espressioni algebriche che si deducono l'una dall'altra meddiante le regole della tabella precedente.

6.3.1 Esempio di espressioni duali

$$\begin{array}{ll} A + (B \cdot C) & A \cdot (B + C) \\ A \cdot \bar{B} + \bar{A} \cdot B & (A + \bar{B}) \cdot (\bar{A} + B) \end{array}$$

Il fatto che gli assiomi di un'algebra di Boole si raggruppino in coppie duali costituisce già una particolarità interessante. Ma più interessante è la considerazione che se ne trae sotto la forma detta del **principio di dualità**. Si consideeri un terorema valido su un'algebra di Boole; per dimostrarlo si deve applicare una successione di operazioni che, in ultima analisi, è una applicazione, in tappe successive, degli assiomi. Se ora si sostituisce ad ognuna di queste operazioni intermedie l'0perazione duale, si otterrà, ovviamente, il risultato duale; si potrà quindi fare a meno di dimostrare il duale di un teorema.

Il principio di dualità si può così enunciare:

A ogni teorema, vero in un'algebra di Boole, corrisponde un teorema duale che è altrettanto vero.

Data una espressione E si indicherà con E l'espressione duale.

6.4 Esempi di algebra di Boole

A	B	A+B	A*B	$\neg A$	$\neg B$
0	0	0	0	1	1
0	1	1	0	1	0
1	0	1	0	0	1
1	1	1	1	0	0

6.4.1 Algebra a due elementi

Si consideri un insieme I a due elementi: siano 0 ed 1; definiamo su questo insieme $0, 1$ 3 operazioni, date dalla tavola 6.1.

L'insieme $0, 1$ fornito delle operazioni $+$, $*$, neg ha una struttura algebrica booleana:

$$B =< 0, 1, +, *, neg, 0, 1 >$$

Per giustificare tale asserzione, è sufficiente dimostrare che gli assiomi $Ai.j$ sono soddisfatti.

Cominciamo col verificare gli assiomi A4.1 e A4.2.

In sede di definizione dell'algebra di Boole, si era detto che l'insieme E doveva essere composto da almeno due elementi, e che tali elementi devevano essere 0 e 1; per cui il nostro insieme I soddisfa questa prima proprietà.

Passiamo a verificare ora la A4.1:

$$A4.1 \qquad A + 0 = 0 + A = A$$

cioè, quale che sia l'elemento $A \in E$, associato con l'elemento $0 \in E$ mediante l'operatore $(+)$, il risultato di tale applicazione deve essere ancora A. E questo si verifica puntualmente nel nostro insieme I; verifichiamo l'assioma A4.1 prima con l'elemento $1 \in I$ e quindi con l'elemento $0 \in I$:

$$A4.1 \qquad 1 + 0 = 1 \qquad 0 + 0 = 0$$

abbiamo cioè riottenuto gli elementi 1 e 0 sfruttando la definizione dell'0perazione $(+)$ (vedi tabella 6.1a).

Analogo discorso si farà per l'assioma A4.2:

$$A4.2 \qquad A \cdot 1 = 1 \cdot A = A$$

usufruendo della tabella 6.1b, che riguarda l'operatore $(\cdot)$ verifichiamo tale assioma per l'insieme I.

$$0 \cdot 1 = 0 \qquad 1 \cdot 1 = 1$$

Per quanto riguarda l'assioma A5:

$$A5.1 \quad A + \bar{A} = 1 \qquad A5.2 \quad A \cdot \bar{A} = 0$$

usufruendo della tabella 6.1, possiamo facilmente verificarlo ponendo $A = 1$, $A = 0$ e viceversa:

$$A5.1 \quad \begin{cases} 1 + \bar{1} = 1 \\ 0 + \bar{0} = 1 \end{cases} \qquad A5.2 \quad \begin{cases} 1 \cdot \bar{1} = 0 \\ 0 \cdot \bar{0} = 0 \end{cases}$$

Lasciamo come esercizioal lettore la verifica dei restanti assiomi.

6.4.2 Algebra delle classi

Un secondo esempio di algebra di Boole è quello costituito dall'insieme di tutti i sottoinsiemi di un dato insieme M quando per $\cup$, $\cap$ e $-$, si prendano rispettivasmente le ordinarie operazioni di **riunione**, **intersezione** e **complementazione** tra insiemi, e perle costanti 0 e 1 rispettivamente l'insieme vuoto Φ e l'insieme delle parti I_M:

$$B_M =< I_M, \cap, \cup, ^-, \phi, I_M >$$

Per vedere che gli assiomi $Ai.j$ sono verificati è sufficiente considerare le relazioni incontrate nel capitolo 5.

Giova rilevare che M non è necessariamente finito; se lo è, se contiene cioè n ($>$2) elementi, l'algebra di Boole B_M ne contiene 2^n.

6.4.3 Algebra delle proposizioni

Consideriamo un insieme L di proposizioni, fornito delle operazioni $\wedge, \vee, \sim$ tale che se $P, Q \in L$, si abbia $\sim P \in L$, $P \wedge Q \in L$, $P \vee Q \in L$. In queste condizioni gli assiomi $Ai.j$ prenderanno la forma delle relazioni $(L_1 ... L_1 0)$: l'insieme L delle proposizioni, con le operazioni $\wedge, \vee, \sim$ è un'algebra di Boole:

$$B_L =< L, \vee, \wedge, \sim, P, P >$$

L'algebra delle proposizioni ha l'implicazione reciproca ($\leftrightarrow$) come relazione di equivalenza.

6.4.4 Algebra dei vettori a n dimensioni

Si consideri l'insieme dei vettori binari a n dimensioni. Un vettore binario (matrice riga) X è un insieme ordinato di variabili (componenti) capaci di assumere uno dei valori:0 e 1:

$$X = (x_1, x_2, ..., x_i..., x_n) = (x_i) \quad x_i \in 0, 1 \quad (i = 1...n)$$

Questo insieme comprende:

$$\underbrace{2 \cdot 2 \cdot 2... \cdot 2}_{n \ fattori} = 2^n \quad elementi$$

Si possono introdurre le operazioni seguenti:
1) Prodotto
Dati due vettori X e Y

$$X = (x_1, x_2...., x - n) = (x_i)$$
$$Y = (y_1, y_2...., y_n) = (y_i)$$

si chiama prodotto booleano di X e Y, denotato con $X \cdot Y$, il vettore

$$X \cdot Y = (x_1 \cdot y_1, x_2 \cdot y_2....x_i \cdot y_i....x_n \cdot y_n) = (x_i \cdot y_i)$$

2) Somma
Dati due vettori X e Y definiti come sopra, si chiama somma booleana di X e Y il vettore:

$$X + Y = (x < -1 + y_1, ...x_i + y_i, ...x_n + y_n) = (x_i + y_i)$$

3) Complemento
Dato un vettore $X = (x_1, .., x_i, ..., x_n)$ il complemento di X è il vettore:

$$\overline{X} = (\overline{x_1}, ...\overline{x_i}, ...\overline{x_n}) = (\overline{x_i})$$

Le operazioni $+$, $.$, $-$, sulle variabili x, y sono quelle definite nell'algebra a due elementi.
4) Vettore nullo
Per definizione esso è un vettore formato unicamente da zeri:

$$0 = (0, 0, ..., 0) \qquad (x_i = 0 \quad i = 1, ...n)$$

5) Vettore unità
È il vettore formato unicamente da 1:

$$1 = (1_1, 1_2, ...1_i, ...1_) \qquad x_i = 1 \qquad i = (1....n)$$

In queste condizioni l'insieme dei vettori binari ad n dimensioni, fornito di queste tre operazioni, è un'algebra di Boole a 2^n elementi, con 0 e 1 per elementi costanti:

$$B =< x, +, \cdot, -, 0, 1 >$$

Si può dimostrare l'asserto facendo vedere che gli assiomi A1.1 sono verificati per i vettori si troverà ogni volta che la dimostrazione si riporta alle componenti. Ora, per quest'ultime, gli assiomi sono verificati in quanto si tratta di un'algebra di Boole a due elementi.

Si verifichi per esempio l'assioma A1.1. Si ha:

$$(X + Y) + Z = X + (Y + Z)$$

ma ciò equivale a scrivere:

$$[(x_i + y_i) + z_i = x_i + (y_i + z_i)] \longleftrightarrow [i(i = 1, 2, 3..., n)(x_i + y_i)z_i = x_i + (y_i + z_i)]$$

Per ogni I quest'ultima relazione è vera in quanto abbiamo visto che per le componenti gli assiomi sono verificati (algebra a due elementi).

Per l'equivalenza, discende che anche la proposizione vettoriale è vera.

L'assioma (A1.1) è dunque verificato.

In maniera analoga si dimostrano gli alti assiomi.

6.5 Variabili ed espressioni booleane

Variabile booleana: è una variabile i cui valori appartengono ad una algebra di boole. Le lettere che abbiamo usato nelle definizioni degli assiomi e dei teoremi fino ad ora, rappresentano tali variabili. Si dirà anche che l'algebra considerata costituisce il dominio della variabile.

Espressione booleana: si chiamerà espressione booleana, o forma booleana, ogni espressione algebrica formata, a partire dalle variabili e dalle costanti appartenenti ad un'algebra booleana, mediante un numero finito di applicazioni delle operazioni $(+, \cdot, ^-)$.

Più precisamente si può dire:

1. 0 e 1 sono espressioni
2. ogni variabile X è una espressione
3. se A è una espresssione, anche $\bar{A}$ lo è
4. se A e B sono espressioni, anche $A + B$ e $A * B$ lo sono
5. un'espressione può essere ottenuta applicando le regole 1, 2, 3 e 4 un numero finito di volte.

6.5.1 Valore di un'espressione booleana

Funzioni generate da una espressione.

Data una espressione booleana ad n variabili, e sia

$$G(x_1, x_2.....x_n)$$

se si sostituiscono le variabili con degli elementi particolari di E, che è l'insieme sostegno, si ottiene come risultato dell'espressione un elemento di E.

Si chiama **assegnazione** (dei valori alle variabili) ogni combinazione $(a_1, a_2...a_n)$ di valori particolari attribuiti alle variabili $(x_1.x_2...x_n)$. L'elemento ottenuto, calcolando la G in corrispondenza dell'assegnazione $(a_1, a_2...a_n)$, è il valore dell'espressione a tale assegnazione.

Sia data ad esempio l'espressione:

$$G(A, B, C) = A \cdot B + A \cdot C + B \cdot C + \bar{A} \cdot \bar{B} \cdot \bar{C}$$

il suo valore per la combinazione (1,0,1) (cioè A=1, B=0, C=1) sarà:

$$G(1, 0, 1) = 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 \cdot 0 = 1$$

Se si calcolano i valori dell'espressione in corrispondenza a tutte le possibili combinazioni dei valori delle variabili, si definisce una funzione $f_G(x_1...x_n)$ di E^n in E. Si dirà allora che l'espressione G genera la funzione in questione e, inversamente, che G è un'espressione o una rappresentanza di f_G.

Consideriamo ad esempio la funzione "Somma di due variabili" che, ad ogni coppia di valori, associa la loro somma; l'espressione sarà:

$$S(A, B) = A + B$$

avremo:

$$S80, 0) = 0 \quad S(0, 1) = 1 \quad S(1, 0) = 1 \quad S(1, 1) = 1$$

in tal modo abbiamo definito la funzione somma; cioè la corrispondenza che, mediante $S(A, B)$, viene generata tra E^2 ed E.

Vi è inoltre da notare che un'espressione definisce un'unica funzione mentre, per una funzione data di E^n in E, esistono infinite (a causa, ad esempio, della legge di idempotenza) espressioni che la generano. Così nell'algebra

$$B =< |0, 1|, +, \cdot, \bar{}, 0, 1 >$$

le espressioni

$$A \cdot \bar{B} + \bar{A} \cdot B \qquad A \cdot \bar{B} + \bar{A} \cdot B + \bar{A} \cdot B \qquad \overline{(A + \bar{B})(\bar{A} + B)}$$

rappresentano la stessa funzione definita dalla tabella:

A	B	$f(A, B)$
0	0	0
0	1	1
1	0	1
1	1	0

6.5.2 Uguaglianza di 2 espressioni booleane

Si dice che 2 espressioni sono uguali (o equivalenti) se esse assumono gli stessi valori per ogni assegnazione di valori delle vasrabili, cioè se esse generano la stessa funzione.

L'uguaglianza di due espressioni G e G' sarà rappresentata dal segno $=$. Si può verificare che essa è una relazione di equivalenza nel senso usuale.

6.6 Forme canoniche. Teorema di Shannon

Come si è detto precedentemente, una funzione booleana può essere rappresentata da infinite espressioni algebriche; tra tutte queste però, ve ne sono due particolari che prendono il nome di forme canoniche: Per giungere alla definizione di tali forme, occorre enunciare il seguente teorema:

6.6.1 Teorema di Shannon per espressioni a una variabile

Ogni espressione booleana di una variabile si può sempre esprimere medfiante una espressione del tipo:

$$F(x) = F(1) * x + F(0) * \bar{x} \qquad (\sigma)$$

dove con $F(1)$ e $F(0)$ s'intende la funzione calcolata per i valori 1 e 0 della variabile.

Verifichiamo la validità di tale teorema perle funzioni base:

a) Funzioni costanti: $F(x) = A$, in tal caso l'espressione è indipendente dal valore della variabile; si ha cioè:

$$F(x) = F(1) = F(0) = A$$

per cui applicando gli assiomi visti nel paragrafo 6.1 si può scrivere:

$$F(x) = A = A \cdot 1 = A(x + \bar{x}) = Ax + a\bar{x} = F(1)x + F(0)\bar{x}$$

b) La funzione assume il valore della variabile X: P(x)=x; avremo:

$$P(1) = 1 \qquad P(0) = 0$$

per cui:

$$P(x) = x = x + 0 = x + 0 \cdot \bar{x} = 1 \cdot x + 0 \cdot \bar{x} = P(1)x + P(0)\bar{x}$$

c) La funzione assume il valore della variabile complimentata: $G(x) = \bar{x}$ avremo:

$$G(1) = 0 \qquad G(0) = 1$$

per cui:

$$G(x) = \bar{x} = 0 + \bar{x} = 0 \cdot x + \bar{x} = 0 \cdot x + 1 \cdot \bar{x} = G(1)x + G(0)\bar{x}$$

Tenendo presente che ogni espressione booleana non sarà altro che una combinazione delle tre espressioni **a), b), c),** si può dimostrare la validità del teorema di **Shannon** per ogni espressione di una funzione ad una variabile. Per ottenere ciò è sufficiente far vedere che: date due funzioni $F(x)$ e $G(x)$ verificanti la (σ), allora anche il loro **prodotto**, la loro **somma**, ed il **complemento** di una di esse la verificano.

d) Sia $H(x) = F(x) * G(x)$ e quindi $H(1) = F(1)G(1)$ e $H(0) = F(0)G(0)$ applicando la (σ) su $F(x)$ e $G(x)$ otterremo:

$$H(x) = [F(1)x + G(1)\bar{x}] \cdot [G(0)x + G(0)\bar{x}]$$

e) Sia $K(x) = F(x) + G(x)$ quindi

$$K(1) = F(1) + G(1) \qquad e \qquad K(0) = F(0) + G(0)$$

applicando la (σ) su $F(x)$ e $G(x)$ otterremo:

$$K(x) = [F(1)x + F(0)\bar{x}] + [G(1)x + G(0)\bar{x}] =$$
$$[F(1) + G(1)]x + [F(0) + G(0)]\bar{x} = K(1)x + K(0)\bar{x}$$

f) Sia $N(x) = \overline{F(x)}$ e quindi

$$N(1) = \overline{F(1)} \qquad e \qquad N(0) = \overline{F(0)}$$

applicando la (σ) su $F(x)$ e per il primo e secondo teorema di Morgan avremo:

$$\begin{aligned}
N(x) &= \overline{F(1)x + F(0)\bar{x} =} \\
&= \overline{F(1)x} \cdot \overline{F(0)\bar{x}} = [\overline{F(1)} + \bar{x}] \cdot [\overline{F(0)} + \bar{\bar{x}}] = \\
&= \overline{F(1)} \cdot \overline{F(0)} + \overline{F(0)}\bar{x} + \overline{F(1)}x + \bar{X}x = \\
&= [\overline{F(1)} \cdot \overline{F(0)}] \cdot 1 + \overline{F(0)}\bar{x} + \overline{F(1)}x = \\
&= [\overline{F(1)} \cdot \overline{F(0)}](x + \bar{x}) + \overline{F(0)}\bar{x} + \overline{F(1)}x = \\
&= \overline{F(1)} \cdot \overline{F(0)} \cdot x + \overline{F(1)} \cdot \overline{F(0)} \cdot \bar{x} + \overline{F(0)} \cdot \bar{x} + \overline{F(1)} \cdot x = \\
&= [\overline{F(1)} \cdot \overline{F(0)} + \overline{F(1)}] \cdot X] + [\overline{F(1)} \cdot \overline{F(0)} + \overline{F(0)}] \cdot \bar{x} = \\
&= \overline{F(1)} \cdot x + \overline{F(0)} \cdot \bar{x} = N(1) \cdot x + N(0) \cdot \bar{x}
\end{aligned}$$

Abbiamo così dimostrato che le operazioni, in una espressione di una funzione booleana, conservano la proprietà di **Shannon** e ciò ci permette sempre di ridurre l'espressione ad una forma lineare ed omogenea nelle variabili x e $\bar{x}$"':

$$f(x) = Ax + B\bar{x} \qquad \text{(dove } A = F(1) \text{ e } B = F(0))$$

nella quale compaiono solo due costanti $F(1)$, $F(0)$ facilmente determinabili data l'espressione della funzione.

6.6.2 Teorema di Shannon per espressioni a n variabili

È possibile enunciare il teorema di **Shannon** anche per espressioni ad n variabili: ma prima di fornire tale enunciato, esponiamo il procedimento con cui si giunge a tale generalizzazione del teorema.

Si consideri una espressione a n variabili:

$$F(x_1, x_2, ..., x_n$$

e si supponga di fissare (cioè considerare costanti) $n - 1$ variabili: l'espressione F è allora una espressione ad una variabile, che gode quindi della proprietà di **Shannon** e può, di conseguenza, essere espressa nella forma:

$$F(x_1, x_2, ..., x_n) = F(1, x_2, x_3, ..., x_n)x_1 + F(0, x_2, x_3, ..., x_n)\bar{x_1}$$

I termini $F(1, x_2, ..., x_n)$ e $F(0, x_2, ..., x_n$ sono, non considerando più $x_2, ..., x_n$ costanti, delle espressioni ad $n-1$ variabili, nelle quuali noi possiamo di nuovo fissare momentaneamenten $n-2$ variabili, per esempio le ultime $n-2$, ed applicare ancora alla variabile n_2 il teorema di **Shannon**. Da cui:

$$\begin{aligned} F(x_1, x_2, ..., x_n) = [F(1, 1, x_3, ..., x_n)x_2 + F(1, 0, x_3, ..., x_n)\bar{x_2}] \cdot x_1 + \\ [F(0, 1, x_3, ..., x_n)x_2 + F(0, 0, x_3, ..., x_n)\bar{x_2}] \cdot \bar{x_1} = \\ F(1, 1, x_3, ..., x_n)x_1x_2 + F(1, 0, x_3, ..., x_n)x_1\bar{x_2} + \\ + F(0, 1, x_3, ..., x_n)\bar{x}_1x_2 + F(0, 0, x_3, ..., x_n)\bar{x}_1\bar{x_2} \end{aligned}$$

Si può continuare ad applicare il teorema di **Shannon** fino all'esaurimento delle variabili; infine si otterrà la seguente espressione:

$$F(x_1, x_2, ..., x_n) = \sum_{e=0,0,..,0}^{e=1,1,..,1} F(e_1, e_2, ..., e_n)e_1^{e_1}e_2^{e_2}...e_n^{e_n}$$

con la convenzione:

$$x_i^{e_i} = \begin{cases} x_i, & amp; \text{quando} \quad e_i = 1 \\ \bar{x}, & amp; \text{quando} \quad e_i = 0 \end{cases}$$

ed $e = (e_1, e_2, ..., ee_n)$ vettore binario a n dimensioni.

Per scrivere in maniera più compatta, si può pensare di considerare **"e"** come la rappresentazione binaria di un numero.

Da ciò la notazione abbreviata:

$$(\sigma') \qquad F(x_1, x_2, ..., x_n) = \sum_{e=0}^{e=2^n-1} F(e)x_1^{e_1}x_2^{e_2}...x_n^{e_n}$$

e ora possiamo enunciare il teorema di **Shannon** per espressioni a n variabili:
ogni funzione booleana di n variabili, si può sempre esprimere mediante una espressione del tipo (σ').

La forma algebrica del secondo membro di (σ') porta il nome di **forma canonica disgiunta**, volendo indicare con questa terminologia, cher essa è una somma di prodotti.

In essa compaiono i 2^n possibili coefficienti $F(e)$ ed altrettanti monomi della forma:

$$(\alpha) \qquad e_1^{e_1}x_2^{e_2}...e_i^{e_i}...x_n^{e_n}$$

Una espressione booleana a n variabili è quindi conosciuta quando si hanno i 2^n coefficienti di $F(e)$.

Un prodotto della forma (α) si chiama **prodotto fondamentale**.

6.6.3 Espressioni duali

Applicando il principio di dualità si deducono le formule seguenti:

$$F(x) = [F(1) + \bar{x}] \cdot [F(0) + x](teoremadiShannonduale)$$

da cui:

$$F(x_1, x_2, ..., x_i, ..., x_n) = \prod_{e=0,0,..,0}^{e=1,1,..,1} [F(\bar{e_1}, .., \bar{e_n}) + x_1^{e_1} + x_2^{e_2} + .. + x_n^{e_n}]$$

che prende il nome di **forma congiuntiva**.

Essa è un prodotto di somme. Le somme della forma:

$$x_1^{e_1} + x_2^{e_2} + ... + x_i^{e_i} + ... + x_n^{e_n}$$

sono chiamate somme **somme fondamentali**.

In maniera più compatta si scriverà:

$$F(x_1, x_2, ..., x_i, ..., x_n) = \prod_{e=0}^{e=2^n-1} [F(2^n - 1 - e) + x_1^{e_1} + x_2^{e_2} + .. + x_n^{e_n}]$$

dove $2^n - 1 - e$ = complemento ad 1 di e.

6.7 Metodo pratico del calcolo di un complemento

Consideriamo l'espressione $E(x_1, x_2, ...x_n)$. Per calcolare il complemento non è necessario applicare passo passo i teoremi di De Morgan: è sufficiente sostituire l'espressione $E(x_1, x_2, ...x_n)$ con l'espressione $\overline{E(x_1, x_2, ...x_n)}$, ottenuta con le regole seguenti:

Si noterà che questa tabella è differente, nelle ultime righe, da quella che definisce la regola di dualizzazione:

6.7.1 Esempio

$$F(A, B, C) = \overline{A \cdot \bar{B} + \bar{C} \cdot \bar{D} + \bar{A} \cdot \bar{B}} + A \cdot B + B \cdot C$$

Per definizione si ha:

$$F(A, B, C) = \overline{\overline{A \cdot \bar{B} + \bar{C} \cdot \bar{D} + \bar{A} \cdot \bar{B}} + A \cdot B + B \cdot C}$$

Per semplificare tale espressione si può applicare il teorema di **De Morgan** oppure la regola sopra enunciata; applichiamo entrambe, e confrontiamo l'uguaglianza dei risultati.

Primo metodo

$$\begin{aligned} \overline{F} = \overline{\overline{A \cdot \bar{B} + \bar{C} \cdot \bar{D} + \bar{A} \cdot B}} \cdot \overline{A \cdot B + B \cdot C} = \\ (A \cdot \bar{B} + \bar{C} \cdot \bar{D} + \bar{A} \cdot B) \cdot (\overline{A \cdot B} \cdot \overline{B \cdot C}) = \\ (A \cdot \bar{B} + \bar{C} \cdot \bar{D} + \bar{A} \cdot B) \cdot (\bar{A} + \bar{B}) \cdot (\bar{B} + \bar{C}) \end{aligned}$$

Secondo metodo

$$(A \cdot \bar{B} + \bar{C} \cdot \bar{D} + \bar{A} \cdot B) \cdot (\bar{A} + \bar{B}) \cdot (\bar{B} + \bar{C})$$

come si vede questo secondo metodo è molto più rapido.

6.8 Riduzione ad un livello di complementazione in una espressione

Data una espressione che comporta più livelli di complementazione sovrapposti, come ad esempio:

$$F = \overline{\bar{A} + B + \overline{C + D} + E}$$

è sempre possibile ricondurre una tale espressione ad una forma dove le complementazioni appaiono, al massimo, ad un livello.

Che ciò sia sempre possibile è una conseguenza dell'esistenza delle forme canoniche.

Comunque, applicando ripetutamente i teoremi di **De Morgan**, si può verificare tale affermazione senza dover ricorrere alle forme canoniche.

Per l'espressione precedente si avrà:

$$F = \overline{(\bar{A} + B)} \cdot \overline{\overline{(C + D)} + E} = (A \cdot \bar{B}) \cdot (C + D) \cdot \bar{E}$$

cioè si è ottenuta una espressione dove gli elementi sono complimentati al massimo una volta.

Tale espressione si potrà sviluppare in somma di prodotti:

$$F = A \cdot \bar{B} \cdot C \cdot \bar{E} + A \cdot \bar{B} \cdot D \cdot \bar{E}$$

6.9 Relazione tra espressione duale e complemento

Data una espressione di una funzione $F(x_1, x_2, ..., x_n)$ le tavole delle regole di dualizzazione e di complementazione permettono di dedurre due espressioni associate alla funzione data:

$$\begin{array}{ll} \sim F(x_1, x_2, ...x_n) & \text{(duale)} \\ \bar{F}(x_1, x_2, ..., x_n) & \text{(complemento)} \end{array}$$

La relazione che lega queste due funzioni è

$$\sim F(x_1, x_2, ..., x_n) =) \bar{F}(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n) \qquad (\beta)$$

i procedimenti per costruire le espressioni $F, simF, \bar{F}$ possono essere riassunte nella tabella nella pagina seguente.

F	eq F	$\bar{\bar{F}}$
0	1	1
1	0	0
A	$A = (\bar{\bar{A}})$	$\bar{A} = (\bar{A})$
$\bar{A}$	$\bar{A} = (\bar{\bar{\bar{A}}})$	$A = (\bar{\bar{A}})$
A+B	$A * B = \overline{\bar{A} + \bar{B}}$	$\bar{A} * \bar{B} = \overline{A + B}$
A*B	A+B=A*B	$\bar{A} + \bar{B} = \overline{A * B}$

F è costituita da un numero finito di operazioni $+$, $*$, $-$, a partire dalle variabili e dalle costanti 0 ed 1; e si vede che applicando passo passo la corrispondenza della tabella, la relazione (β) è verificata.

Fonte del testo: https://it.wikibooks.org/w/index.php?title=Algebre_booleane_e_progetto_logico_dei_calcolatori_digitali/Algebre_booleane&oldid=360937

7

Funzioni booleane

7.1 Definizioni

Si definisce variabile binaria, quella variabile che ha per dominio l'insieme $\{0,1\}$ composto di 2 elementi. Il prodotto cartesiano di tale insieme per se stesso, cioè l'insieme delle coppie ordinate (a_1, a_2) di elementi in $\{0,1\}$, verrà indicato con $\{0,1\}^2 = \{0,1\} \times \{0,1\}$.

Iterando il ragionamento, l'insieme $\{0,1\}^n = \{0,1\} \times \{0,1\} \times ... \{0,1\}$ sarà composto dalle n-ple ordinate $a_1, a_2 ... a_n)$, di elementi in $\{0,1\}$. A questo punto è possibile definire la funzione booleana di n variabili(o anche funzione binaria, o funzione di commutazione) come una applicazione di $\{0,1\}^n$ in $\{0,1\}$.

Abbiamo già detto che una funzione booleana può essere espressa o mediante una classe di espressioni booleane equivalenti, oppure mediante una tabella della verità nella quale in corrispondenza di ogni combinazione delle n variabili deve scrivere il valore associato della funzione. Se si assume come convenzione di scrivere le combinazioni delle n variabili in ordine crescente, cioè in modo tale che leggendole come numeri binari, esse rappresentino la sequenza di numeri da 0 a 2^n, allora non sarà necessario scrivere tutta la tabella, ma risulterà sufficiente scrivere solo i valori della funzione.

$x1$	$x2$	$x3$	$x4$	$x5$	f
0	0	0	0	0	1
0	0	0	0	0	1
0	0	1	1	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1

Le tabelle della verità possono essere anche di tipo rettangolare, e cioè le n variabili sono ripartite in due insiemi disgiunti $(x_1, ..., x_s)$ e $(x_{s+1}, ..., x_n)$

A ciascuno dei due insiemi si assegna una delle due dimensioni del rettangolo e su di essa l'insieme di variabili assume tutte le possibili combinazioni: a ogni combinazione corrisponde una colonna (o una riga).

Spieghiamo con un esempio il funzionamento del nuovo tipo di tabella, paragonandolo alla tabella usata fino ad ora nella quale però, per brevità, non prenderemo in esame le combinazioni che corrispondono ad un valore nullo della funzione.

Sia $f(x_1, x_2, x_4, x_5)$ definita dalla tabella (A) qui sotto:

	x3	0	1	0	1	0	1	0	1
	x2	0	0	1	1	0	0	1	1
	x1	0	0	0	0	1	1	1	1
x1	x2								
0	0								
0	1								
1	0								
1	1								

le combinazioni relative a x1,x2,x3 vengono lette dal basso in alto.

Volendo riscrivere la tabella, questa volta in forma rettangolare procediamo prima nel seguente modo:

Consideriamo, ad esempio, la 5ª colonna. Ad essa corrisponde l'assegnazione $(x_1, x_2, x_3) = (100)$ ed alla 2ª riga corrisponde l'assegnazione $(x_4 x_5) = (01)$. Se ora consideriamo contemporaneamente le due assegnazioni, esse corrispondono all'unica $(x_1, x_2, x_3, x_4, x_5) = (10001)$. A tale combinazione, nella tabella (A), corrisponde il valore 1 (della funzione); Nella tabella b (a fianco), allora, scriveremo all'intersezione della 5ª colonna con la 2ª riga il valore 1.

	x3	0	1	0	1	0	1	0	1
	x2	0	0	1	1	0	0	1	1
	x1	0	0	0	0	1	1	1	1
x3	x4								
0	0	1	0	0	1	0	1	0	1
0	1	1	0	1	0	1	0	1	0
1	0	0	1	1	1	1	0	0	1
1	1	0	0	0	0	1	1	1	0

La tabella completa avrà la forma della tabella b_2 accanto:

È sotto questa forma che utilizzeremo soprattutto le tabelle della verità: messa infatti sotto questa forma conveniente (diagramma di Karnaugh), una simile tabella è un mezzo per la semplificazione delle espressioni algebriche.

Inoltre questa forma serve ogni qualvolta si vuole mettere in rilievo il ruolo di una certa variabile; in tal caso le si attribuirà un lato della tabella.

7.2 Altri modi di definire una funzione booleana

7.2.1 Vettore caratteristico

La colonna di destra della tavola della verità, del 1° tipo, è un vettore binario.

$$V_f = 10100110$$

N_0	X_1	X_2	X_3	$f(X_1, X_2, X_3)$	
0	0	0	0	1	V_0
0	0	0	1	0	V_1
2	0	1	0	1	V_2
3	0	1	1	0	V_3
4	1	0	0	0	V_4
5	1	0	1	1	V_5
6	1	1	0	1	V_6
7	1	1	1	0	V_7

Se si conservano le convenzioni fatte sulle combinazioni delle variabili, la conoscenza di questo vettore è sufficiente a definire la funzione. Chiameremo tale vettore **vettore caratteristico** della funzione f e lo denoteremo con:

$$V_f = (V_0, V_1, ...V_k, ...V_{2^n-1})$$

dove V_k è il valore della funzione associata a quella combinazione binaria delle variabili, che rappresenta il numero k (esempio figura accanto).

7.2.2 Rappresentazione decimale

Prima rappresentazione

Il Vettore V_f costituisce la rappresentazione binaria di un numero intero compreso tra 0 e 2^n.

Si può allora definire la funzione mediante l'equivalenza decimale di questo numero:

$$N_{10}^f = \sum_{i=0}^{2^n-1} v_i * 2^n - 1 - i$$

Esempio:

$$V_f = (10100110)$$

$$N_{10}^f = 2^7 + 2^5 + 2^2 + 2^1 = 166$$

Seconda rappresentazione

Forniamo una 2ª rappresentazione del vettore caratteristico: rappresentazione molto usata anche se meno compatta della prima

Se si esamina l'esempio 1°, si vede che il vettore caratteristico assume il valore 1 in corrispondenza a quelle combinazioni che rappresentano i numeri decimali **0,2,5,6**. Allora si può rappresentare il vettore caratteristico, fornendo quest'insieme di valori decimali.

Esempio:

$$V_f = (10100110) = (0, 2, 5, 6)$$

1^a *rappr.* 2^a *rappr.*

7.3 Operazioni sulle funzioni booleane

Usando le tavole della verità, che ci permettono di definire le funzioni booleane, possiamo ora definire le operazioni di **somma**, **prodotto** e **complementazione delle funzioni**.

X_1	X_2	f	g	$f+g$
0	0	0	0	0
0	1	0	1	1
1	0	1	1	1
1	1	1	0	1

7.3.1 Somma di 2 funzioni f+g

Date le due funzioni f e G, a n variabili, con i vettori caratteristici V_f e V_g, la funzione **somma di f e g** è quella il cui vettore caratteristico è $V_f + V_g$ (vedi 6.4.4).

Si ha dunque per definizione:

$$V_{f+g} = V_f + V_g$$

7.3.2 Prodotto di 2 funzioni: f*g

X_1	X_2	f	g	$f*g$
0	0	0	0	0
0	1	0	1	0
1	0	1	1	1
1	1	1	0	0

In maniera analoga, il prodotto di f e g è la funzione di vettore caratteristico $V_f * V_g$:

$$V_{f*g} = V_f * V_g$$

7.3.3 Complemento di una funzione: $\neg f$

x_1	x_2	f	not f
0	0	0	1
0	1	0	1
1	0	1	0
1	1	0	1

È la funzione notata $\neg f$ e definita mediante $\neg V_f$

Per definizione si ha quindi:

$$\neg V_f = V_{\neg f}$$

7.3.4 Funzione identicamente nulla: (0)

È la funzione il cui vettore caratteristico è il vettore nullo.

7.3.5 Funzione identicamente uguale a 1

È la funzione il cui vettore caratteristico è il vettore (1).

L'insieme delle funzioni booleane a n variabili possiede 2^n elementi distinti.Infatti ogni vettore caratteristico è formato da una sequenza di 2^n elementi: l'insieme delle funzioni sarà dato da tutte le possibili combinazioni (disposizioni con ripetizione) di questi 2^n elementi; per cui avremo 2^{2^n} vettori distinti.

È mediante l'aiuto dei vettori caratteristici che noi abbiamo definito sull'insieme delle funzioni booleane le operazioni $+$, $*$, $^-$.

Questi ultimi formano un'algebra di Boole a $2^N = 2^{2^n}$ elementi e quindi segue che l'insieme delle funzioni di n variabili, munite di queste operazioni, formano ugualmente un'algebra di Boole a 2^N elementi.

7.4 Espressioni algebriche di una funzione booleana

Esaminiamo ora il seguente problema: data una funzione booleana di n variabili $x_1...x_n$ conosciuta mediante la sua tavola della verità, trovarne una rappresentazione algebrica.

Per fare ciò definiamo alcune funzioni di n variabili che ci saranno utili in seguito.

7.4.1 Prodotto fondamentale

Definizione: si chiama prodotto fondamentale o **minterm** a n variabili un prodotto della forma:

$$P = x_1^{e_1} * x_2^{e_2} * ... * x_n^{e_n} = \sum_{i=1}^{i=n} x_i^{e_i}$$

- con $x_i^{e_i} = x_i$ se $e_i = 1$ cioè $x^1 = x$
- con $x_i^{e_i} = \bar{x}_i$ se $e_i = 0$ cioè $x^0 = \bar{x}$

In altri termini un prodotto fondamentale è un prodotto di n variabili (dirette o complamentate) in cui ogni variabile compare una sola volta.

7.4.2 Tavole della verità di un prodotto fondamentale

Affinché il prodotto

$$P = e_1^{e_1} \cdot x_2^{e_2} \cdot .... \cdot x_n^{e_n}$$

sia uguale a 1, è necessario e sufficiente che tutti i suoi fattori siano uguali a 1.

Cioè che si abbia:

$$e_i^{e_i} = 1 \qquad (i = 1, 2.....n)$$

Ora con riguardo a quanto detto nel par.6.6 si ha:

$$e_i^{e_i} \begin{cases} = x_i & amp; \text{se} \quad e_i = 1 \\ = \bar{x}_i & amp; \text{se} \quad e_i = 0 \end{cases}$$

Quindi perché $e_i^{e_i}$=1, è necessario e suffciente che $x_i = e_i$.

X_1	X_2	X_3	P
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Per cui affinché $P = 1$ è necessario e sufficiente che si abbia:

$$x_i = e_i \qquad (i = 1, 2, ..., n)$$

Il prodotto fondamentale P è dunque uguale ad 1 quando le variabili assumono le combinazioni $(e_1, e_2, ...e_n)$; cioè quando si ha che:

$$x_1 = e_1 \quad x_2 = e_2, ...x_n = e_n)$$

Per esempio, il prodotto fondamentale $x_1^1 x_2^0 x_3^1$ assumerà il valore 1 solo per la combinazione (1 0 1); cioè, se esprimiamo tale prodotto sotto forma di tabella, avremo (tabella affiancata):

Diremo che una funzione è generata da un prodotto fondamentale, quando la sua tabella (o il vettore caratteristico è del tipo di cui alla tabella mostrata, e rappresentiamo la funzione con l'espressione:

$$f = P = e_1^{e_1} \cdot ..... \cdot e_n^{e_n}$$

dove $e_1...e_n$ è la combinazione di valori per cui il prodotto fondamentale risulta uguale ad 1.

Tabella 7.1: Tabella A

X_1	X_2	X_3	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	0	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Consideriamo ora la funzione a tre variabili definita dalla tabella A e prendiamo in esame le tre funzioni f_1, f_2, f_3 definite dalla tabella B: ora, confrontando i vettori V_f, V_{f_1}, V_{f_2}, V_{f_3}, è facile verificare che le quattro funzioni soddisfano la seguente relazione:

$$f = f_1 + f_2 + f_3$$

Tabella 7.2: Tabella B

X_1	X_2	X_3	f_1	f_2	f_3
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	0	0	0
1	0	0	0	0	1
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0

dove le f_1, f_2, f_3, sono generate, ciascuna, da un prodotto fondamentale. Indichiamo quest'ultimo prodotto con m_{abc}, dove **abc** è la combinazione binaria che rende uguale ad 1 tale prodotto; cioè avremo:

$$m_{abc} = x_1^a \cdot x_2^b \cdot x_3^c$$

Se ora consideriamo i tre prodotti fondamentali che generano le funzioni f_1, f_2, f_3, e trasformiamo il numero binario **abc** in decimale, avremo le seguenti notazioni:

$$\begin{aligned} m_{001} &= x_1^0 x_2^0 x_3^1 = \bar{x}_1 \bar{x}_2 x_3 = m_1 \qquad (001_2 = 1_{10}) \\ m_{010} &= x_1^0 x_2^1 x_3^0 = \bar{x}_1 x_2 \bar{x}_3 = m_1 \qquad (010_2 = 2_{10}) \\ m_{100} &= x_1^1 x_2^0 x_3^0 = x_1 \bar{x}_2 \bar{x}_3 = m_1 \qquad (100_2 = 4_{10}) \end{aligned}$$

e inoltre, essendo le tre funzioni generate da tali prodotti, si può scrivere:

$$f_1 = m_1 \qquad f_2 = m_2 \qquad f_3 = m_4$$

da cui

$$f = f_1 + f_2 + f_3 = m_1 + m_2 + m_4 = \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 \bar{x}_3$$

È chiaro che il ragionamento effettuato nel caso di tre variabili e su di una funzione particolare, vale anche in generale; e che ogni funzione booleana si può esprimere come somma booleana dei prodotti fondamentali.

Per fare ciò si seguiranno le seguenti regole:

1. si scrivono tutti i prodotti fondamentali corrispondenti alle combinazioni delle variabili per le quali la funzione $f(x_1 ... x_n)$ vale 1.
2. si forma la somma booleana di tutti questi prodotti.

L'espressione così ottenuta, costituisce una rappresentazione algebrica della funzione, chiamata **prima forma canonica** o **forma canonica disgiuntiva**. Si ha:

$$f(x_1, ..x_i, ..x_n) = \sum_{e_1..e_n=0}^{e_1..e_n=2^n-1}$$

La prima forma canonica è unica: il carattere di unicità risulta direttamente dalla corrispondenza biunivoca che esiste tra i prodotti fondamentali componenti la forma canonica e le combinazioni binarie della tabella della verità per le quali la funzione vale 1.

Tabella 7.3: Tabella funzione f

X_1	X_2	X_3	f
0	0	0	0
0	0	0	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Esempio

Si voglia determinare la forma canonica disgiuntiva della funzione f definita dalla tabella relativa.

Per fare ciò, determiniamo prima i tre prodotti fondamentali:

$$m_{001} = m_1 = x_1^0 x_2^0 x_3^1 = \bar{x}_1 \bar{x}_2 x_3$$
$$m_{101} = m_5 = x_1^1 x_2^0 x_3^1 = x_1 \bar{x}_2 x_3$$
$$m_{111} = m_7 = x_1 x_2 x_3$$

da ciò ne discende che la forma canonica disgiuntiva della funzione sarà:

$$f = \bar{x}_1 \bar{x}_2 x_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 x_3$$

7.4.3 Somma fondamentale

Si chiama **somma fondamentale**, o **maxterm** una espressione della forma:

$$S = e_1^{e_1} + ... + x_i e_i ..... + x_n^{e_n} = \sum_{i=0^i=n} x_i^{e_i}$$

dove, per le $x_i^{e_i}$, valgono le convenzioni fatte precedentemente.

In altri termini, una somma fondamentale a n variabili è una somma delle n variabili (dirette o complementate) dove ciascuna delle x_i compare una sola volta.

7.4.4 Tavola della verità di una somma fondamentale di n variabili

Affinché la somma:

$$S = x_1^{e_1} + ... + x_i^{e_i} + ... + x_n^{e_n}$$

sia nulla, è necessario e sufficiente che tutti i suoi termini siano nulli: cioè che si abbia:

$$x_i^{e_i} = 0 \qquad (i = 1, 2, ..., n) \qquad\qquad ovvero\ x_i = \bar{e}_i$$

Dunque, perché S sia nulla, è necessario e sufficiente che la combinazione delle variabili sia identica a quella delle e_i complementate.

Riassumendo: la somma S ha una tavola della verità che è nulla in corrispondenza ad una sola combinazione delle variabili:

$$(\bar{e}_1, \bar{e}_2 ... \bar{e}_n)$$

e vale 1 per tutte le altre.

Inversamente, ogni funzione avente una tavola della verità di questo tipo, cioè nulla per una sola combinazione $(a_1, a_2, ..., a_n)$ delle variabili, potrà essere rappresentata con una somma fondamentale:

$$f = x_1^{\bar{a}_1} + x_2^{\bar{a}_2} + ... + x_n^{\bar{a}_n}$$

cioè se $(a_1 a_2 ... a_n)$ è l'unica combinazione che annulla la f potremo scrivere tale funzione come somma delle x_i aventi, per indice superiore, i valori complementari della **n-pla**.

Esempio

x_1	x_2	x_3	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

1) La somma fondamentale

$$x_1 + \bar{x}_2 + \bar{x}_3 + x_4 = x_1^1 + x_2^0 + x_3^0 + x_4^1$$

è nulla per la combinazione **0110** e per questa soltanto.

2) Si consideri la funzione definita dalla tabella affiancata: essendo **011** l'unica combinazione che annulla la f si può scrivere:

x_1	x_2	x_3	f	f_1	f_2	f_3	$f = f_1 f_2 f_3$
0	0	0	1	1	1	1	1
0	0	1	0	0	1	1	0
0	1	0	0	1	0	1	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	0
1	0	1	1	1	1	1	1
1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1

$$f = x_1^{\bar{0}} + x_2^{\bar{1}} + x_3^{\bar{1}} = x_1 + \bar{x}_2 + \bar{x}_3$$

3) Si considerino le funzioni f, f_1, f_2, f_3 definite dalla tabella sulla destra:

esaminando 9 vettori caratteristici delle funzioni si vede che si può scrivere $f = f_1 f_2 f_3$, dove f_1, f_2, f_3, sono funzioni generate da somme fondamentali associate agli zeri della colonna f

Si ha inoltre che:

$$f_1 = x_1 + x_2 + \bar{x}_3$$
$$f_2 = x_1 + \bar{x}_2 + x_3$$
$$f_3 = \bar{x}_1 + f_2 + x_3$$

da cui

$$f = (x_1 + x_2 + \bar{x}_3) \cdot (x_1 + \bar{x}_2 + x_3) \cdot (\bar{x}_1 + f_2 + x_3)$$

7.4.5 Forma canonica congiuntiva di una funzione a n variabili

Ragionando come nel caso della forma disgiuntiva, si vede che ogni funzione potrà essere espressa come un prodotto di somme fondamentali, utilizzando le regole seguenti:

1. scrivere tutte le somme fondamentali corrispondenti alle combinazioni per le quali la funzione $f = (x_1 ... x_n$ vale zero.
2. scrivere il prodotto di queste somme.

L'espressione così ottenuta, costituisce una rappresentazione algebrica della funzione, chiamata: **seconda forma canonica** o **forma canonica congiuntiva**.

Si scriverà in generale:

$$f(x_1, x_2 .... x_n) = \prod_{e_1..e_n=0}^{e_1..e_n=2^n-1} [f(\bar{e}_1 ... \bar{e}_n) + x_1^{e_1} + ... + x_n^{e_n}]$$

X_1	X_2	X_3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Esempio. Si voglia determinare la forma canonica congiuntiva della funzione tabellata a fianco:

Le somme fondamentali sono:

$$f_1 = x_1 + x_2 + \bar{x}_3$$
$$f_2 = x_1 + \bar{x}_2 + \bar{x}_3$$
$$f_3 = \bar{x}_1 + x_2 + \bar{x}_3$$
$$f_4 = \bar{x}_1 + \bar{x}_2 + \bar{x}_3$$

e la funzione f avrà la seguente forma canonica congiuntiva:

$$f = (x_1 + x_2 + \bar{x}_3) + (x_1 + \bar{x}_2 + \bar{x}_3) + (\bar{x}_1 + x_2 + \bar{x}_3) + (\bar{x}_1 + \bar{x}_2 + \bar{x}_3)$$

La seconda forma canonica, come la prima, è unica. Ciò discende dall'identico processo logico seguito.

Un'altra considerazione è che nella forma canonica disgiuntiva i termini il cui coefficiente risulta essere $f(e_1...e_n)$ scompaiono se f è nulla. Nella forma canonica congiuntiva, i fattori scompaiono dal prodotto quando $f(\bar{e}_1...\bar{e}_n)$ vale1, cioè quando il fattore è della forma

$$1 + x = 1$$

7.4.6 Qualche notazione abbreviata

Prodotti fondamentali

$$\text{si ponga} \qquad m_i = e_1^{e_1} \cdot x_2^{e_2}....x_n^{e_n}$$

dove $i = e_1e_2...e_n$ è un indice espresso nella base decimale e tale che $0 \leq i \leq 2^n - 1$, mentre $e_1e_2....e_n$ è la sua rappresentazione binaria.

$$\text{si ha} \qquad m_i \cdot m_j = 0 \qquad \textit{per } i \neq j$$

in quanto I e **j** sono i numeri che corrispondono all'unica combinazione binaria che rende uguale ad 1 il prodotto fondamentale.

Somme fondamentali

$$\textit{Si ponga} \qquad M_i = x_1^{e_1} + x_2^{e_2} + ... + x_n^{e_n}$$

dove per $i = e_1e_2....e_n$ vale quanto sopra.

$$\textit{Si avra} \qquad M_1 + M_2 = 1 \qquad \textit{per } i \neq j$$

Forma canonica disgiuntiva

$$f = \sum_{i=0}^{i=2^n-1} f(i)m_i$$

Forma canonica congiuntiva

$$f = \prod_{i=0}^{i=2^n-1} f(2^n - 1 - i) + M_i$$

poiché $\bar{e}_1\bar{e}_2....\bar{e}^n = 2^n - 1 - i$ se si pone $i = e_1e_2....e_n$.

7.5 Funzioni duali

Data una funzione $f(x_1, x_2, ..., x_n)$, si chiama funzione duale la funzione $\tilde{f}(x_1, x_2, ...x_n)$ che soddisfa alla relazione

$$\tilde{f}(x_1, x_2, .., x_n) = \overline{f(\bar{x}_1, .., \bar{x}_n)} = f(x_1, ..., x_n)$$

7.5.1 Proprietà duale di una funzione duale

$$\tilde{\tilde{f}} = f$$

infatti

$$\tilde{\tilde{f}} = (\tilde{\bar{f}}) = (\bar{\bar{f}}(\bar{x_1}, ...\bar{x_n})) = f(x_1, .., x_n)$$

7.5.2 Proprietà duale di una somma

$$\sim \sum_i x_i = \overline{\sum_i \bar{x}_i} = \prod_i x_i$$

7.5.3 Proprietà duale di un prodotto

$$\sim \prod_i x_i = \overline{\prod_i \bar{x}_i} = \sum_i x_i$$

7.5.4 Proprietà duale di un complemento

$$\sim (\bar{A}) = \overline{\overline{(\bar{A})}} = \bar{A}$$

7.5.5 Proprietà duale di una funzione costante

Poiché stiamo considerendo un'algebra a due elementi 0 e 1, le funzioni costanti possono assumere solo l'uno o l'altro di questi due valori, per cui se:

$$f(x_1, x_2, ..., x_n) = 0$$

allora si ha che

$$f(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n) = 0$$

in quanto il valore di una funzione costante non dipende dall'assegnazione delle variabili. Risulta allora:

$$\sim f(x_1, ..., x_n) = \bar{f}(\bar{x}_1, ..., \bar{x}_n) = 1$$

Nella stessa maniera si ha che se

$$f(x_1, ..., x_n) = 1$$

allora

$$\sim f(x_1, ..., x_n) = 0$$

Riassumendo possiamo dire che, data una espressione rappresentante una funzione, si ottiene l'espressione della funzione duale sostituendo a + con •, • con + e lasciando le variabili (complementate o no) tali e quali e sostituendo 0 con 1, e 1 con 0; regole che sono le stesse viste nel capitolo 6 in generale.

7.6 Funzioni particolari

Nel capitolo **5** sono state definite le funzioni di AND, OR, $PEIRCE$, $SHEFFER$, di due argomenti; vogliamo ora definire di nuovo tali funzioni attribuendo loro n argomenti. con ciò non intendiamo dire che la singola funzione opera ripetutamente fino all'esaurimento delle variabili, bensì che essa viene considerata una operazione **n-aria** che opera contemporaneamente sulla **n-pla** di argomenti.

Questa osservazione rivela la sua importanza in considerazione del fatto che non tutte le funzioni soprannominate godono della proprietà associativa e quindi, se la funzione non opera contemporaneamente su tutti gli argomenti, otterremmo risultati diversi a seconda delle associazioni fatte, e di conseguenza arbitrari.

7.6.1 Funzione di AND a n variabili

È la funzione che vale

$$f(n) = \begin{cases} 1, \text{ quando } x_1 = x_2 = ... = x_n = 1 \\ 0, \text{ altrimenti} \end{cases}$$

si scrive

$$f = x_1 \cdot x_2 \cdot .... \cdot x_n$$

7.6.2 Funzione di OR a n variabili

È la funzione che vale

$$f(n) = \begin{cases} 1, \text{quando } x_1 = x_2 = ... = x_n = 0 \\ 0, \text{altrimenti} \end{cases}$$

si scrive

$$f = x_1 + x_2 + ... + x_n$$

7.6.3 Funzione di PEIRCE a n variabili o funzione di NOR

È la funzione che vale

$$f(n) = \begin{cases} 1, \text{quando } x_1 = x_2 = ... = x_n = 0 \\ 0, \text{altrimenti} \end{cases}$$

e si scrive

$$f = x_1 \downarrow x_2 \downarrow x_3 ... \downarrow x_n = \downarrow_{i=1}^{i=n} x_i$$

questa funzione è definita anche per $n = 1$ cioè:

$$\downarrow x = x \downarrow = \begin{cases} 1, \text{se } x = 0 \\ 0, \text{se } x = 1 \end{cases} \qquad da\ cui\ \downarrow x = x \downarrow = \bar{x}$$

Per la funzione $PEIRCE$ non vale la legge associativa.

7.6.4 Funzione di $SHEFFER$ a n variabili o funzione di $NAND$

È la funzione che vale

$$f(n) = \begin{cases} 0, \text{quando } x_1 = x_2 = ... = x_n = 1 \\ 1, \text{altrimenti} \end{cases}$$

si scrive

$$f = x_1/x_2/.../x_n = /_{i=1}^{i=n} x_i$$

anche la funzione di $SHEFFER$ è definita se $n = 1$ cioè $/x = x/ = \bar{x}$, e non gode della proprietà associativa.

7.7 Proprietà degli operatori ↓ e /

7.7.1 Espressione degli operarori ↓ e / mediante (+), (·), (-)

Tenendo presente le definizioni ora date, vediamo che la funzione **PERIRCE** vale 1 per una sola combinazione di valori delle variabili, ma allora possiamo facilmente scrivere tale funzione, mediante la **1ª forma canonica**, come il prodotto fondamentale m_0 cioè:

$$\downarrow_{i=0}^{i=n} x_i = \downarrow x_2 \downarrow x_3 \downarrow ... \downarrow x_n = m_0 = x_1^0 x_2^0 .... x_n^0 = \bar{x}_1 \cdot \bar{x}_2 \cdot .... \cdot \bar{x}_n$$

o ancora, tenendo presente le regole della complementazione

$$\downarrow_{i=0}^{=n} x_i = \prod_i \bar{x}_i = \overline{\sum_i x_i}$$

Viceversa, la funzione di $SHEFFER$, si annulla in un solo caso; quindi possiamo esprimerla facilmente, mediante la **2ª forma canonica**, come la somma fondamentale M_0 cioè:

$$/_{i=0}^{i=n} x_i = x_1/x_2/.../x_n = M_{00..0} = M_0 = x_1^0 + x_2^0 + ... + x_n^0 = \sum_i \bar{x}_i$$

o ancora, tenendo presente le regole della complementazione

$$/_{i=0}^{i=n} x_i = \sum_i \bar{x}_i = \overline{\prod_i x_i}$$

7.7.2 Proprietà di dualità

Vedremo ora le proprietà di complementazione, di pseudo associatività e altre, di ↓ e

Consideriamo la funzione di $PEIRCE$ $x_1 \downarrow x_2 \downarrow ... \downarrow x_n$ a n variabili.

Si può scrivere:

$$\overline{\bar{x}_1 \downarrow ... \bar{x}_n \downarrow} = \overline{\prod_i \bar{x}_i} = \overline{\prod_i x_i} = \sum_i \bar{x}_i = x_1/x_2/.../x_n$$

Dunque la funzione duale della funzione di $PEIRCE$ (↓) è la funzione di $SHEFFER$ ().

Viceversa: considerando la funzione di **Sheffer** $x_1/x_2/.../x_n$ a n variabili, si può scrivere:

$$\overline{\bar{x}_1\bar{x}_2/..\bar{x}_n} = \overline{\sum_i \overline{\bar{x}_i}} = \overline{\sum_i x_i} = \prod_i \bar{x}_i = x_1 \downarrow x_2 \downarrow .. \downarrow x_n$$

Quindi la funzione duale di Sheffer **(/)** è la funzione di Peirce (↓).

Consideriamo la funzione di Peirce espressa mediante l'operazione (*), si ha: $\downarrow_i x_i = \prod_i \bar{x}_i$ complementando ambo i membri si ottiene:

$$\overline{\downarrow_i x_i} = \overline{\prod_i \bar{x}_i} = \sum_i x_i \qquad (a)$$

ma la funzione di Sheffer era stata espressa mediante una somma cioè:

$$/_i x_i = \sum_i \bar{x}_i \qquad (b)$$

da cui confrontando la (A) con la **(b)** risulta:

$$\overline{\downarrow_i x_i} = /_i \bar{x}_i$$

Ripetendo il ragionamento sulla funzione di Sheffer, espresso mediante la somma si ottiene:

$$\overline{/_i x_i} = \downarrow_i \bar{x}_i$$

Le ultime due relazioni generalizzano in un certo senso la formula di De Morgan:

$$\overline{\sum_i x_i} = \prod_i \bar{x}_i = \downarrow x_i$$

$$\overline{\prod_i} x_i = \sum_i \bar{x}_i = /x_i$$

Pseudo associatività

Come abbiamo già detto, gli operatori di Sheffer e di Peirce non sono associativi. Consideriamo le espressioni:

$$F = A \downarrow (B \downarrow C)$$

$$F' = (A \downarrow B) \downarrow C$$

si ha che $F \neq F'$, infatti:

$$F = \overline{A + (B \downarrow C)} = \bar{A} \cdot \overline{B \downarrow C} = \bar{A} \cdot \overline{\overline{(B + C)}} = \bar{A}(B + C) = \bar{A}B + \bar{A}C$$

$$F' = \overline{(A \downarrow B) + C} = \overline{A \downarrow B} \cdot \bar{C} = (A + B)\bar{C} = A\bar{C} + B\bar{C}$$

Dunque, per un insieme di dati valori **ABC**, F e F' non sono necessariamente uguali. In altri termini l'operazione ↓ non è associativa. Si vedrà in maniera analoga che l'operazione / non è associativa.

Tuttavia tali operatori godono di una notevole proprietà detta pseudo associatività:

$$A \downarrow B \downarrow C = A \downarrow \overline{(B \downarrow C)} = \overline{(A \downarrow B)} \downarrow C$$

Dimostrazione:

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

Queste proprietà si generalizzano, nel caso di n variabili, come segue:

$$\overline{x_1 \downarrow x_2 \downarrow ... \downarrow x_k} \downarrow k_{k+1} \downarrow ... \downarrow x_n = x_1 \downarrow x_2 ... \overline{x_k \downarrow x_{k+1} ... \downarrow x_n} \qquad (1 < k < n)$$

$$\overline{x_1/x_2/.../x_k}/k_{k+1}/.../x_n = x_1/x_2 ... \overline{x_k/x_{k+1}.../x_n} \qquad (1 < k < n)$$

Dalla definizione dell'**operatore** $\downarrow$ si deducono le seguenti relazioni:

$$\begin{cases} x_1 \downarrow x_2 ... \downarrow x_n \downarrow 0 = x_1 \downarrow x_2 \downarrow ... \downarrow x_n & (n > 1) \\ x_1 \downarrow x_2 \downarrow ... \downarrow x_n \downarrow 1 = 0 \end{cases}$$

$$\begin{cases} x_1/x_2 .../x_n/1 = x_1/x_2/.../x_n & (n > 1) \\ x_1/x_2/.../x_n/0 = 1 \end{cases}$$

$$\begin{cases} 0 \downarrow x = x \downarrow 0 = \bar{x} & (n = 1) \\ 1/x = x/1 = \bar{x} \end{cases}$$

7.7.3 Espressione degli operatori (·),(+),(-)

Mediante gli operatori ($\downarrow$) e (/) (n variabili)

Operazione ($\cdot$)

$$x_1 \downarrow x_2 \downarrow ... \downarrow x_n = \bar{x_1} \cdot \bar{x_2} \cdot ... \cdot \bar{x_n}$$

ponendo $x_i = \bar{A_i}$

$$\prod_i A_i = A_1 \cdot A_2 \cdot ... \cdot \cdot A_n = \bar{A_1} \downarrow \bar{A_2} \downarrow ... \downarrow \bar{A_n} = \downarrow_i \bar{A_i}$$

Operazione (+)

In maniera analoga si ha :

$$x_1/x_2/.../x_n = \bar{x_1} + \bar{x_2} + ... + \bar{x_n}$$

ponendo $x_i = \bar{A_i}$

$$\sum_i A_i = A_1 + A_2 + ... + A_n = \bar{A_1}/\bar{A_2}/.../\bar{A_n} = /_i \bar{A_i}$$

Complemento (-)

È facile verificare dalle definizioni che

$$\bar{A} = \overline{A + A + ... + A} = A \downarrow A \downarrow ... \downarrow A$$

Analogamente:

$$\bar{A} = \overline{A \cdot A \cdot ... \cdot A} = A/A/.../A$$

Inoltre grazie a delle proprietà già viste è possibile esprimere il complemento anche nel modo seguente:

$$\bar{x} = 0 \downarrow x = x \downarrow 0$$
$$\bar{x} = 1/x = x/1$$

Da ciò è facile vedere che:

$$x \downarrow \bar{x} = \overline{x/\bar{x}} = 0$$

7.7.4 Forme canoniche di Sheffer (n variabili)

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

A partire dalle due forme canoniche è facile ottenere delle espressioni utilizzanti ↓ e /, che si possono chiamare forme canoniche di Sheffer.

Consideriamo la funzione F a tre variabili, **A,B,C**, definita dalla tabella collaterale. Per la prima forma canonica abbiamo che:

$$F = \bar{A}BC + \bar{A}\bar{B}C + AB\bar{C}$$

a) possiamo allora scrivere:

$$F = \overline{(\overline{\bar{A}BC})(\overline{\bar{A}\bar{B}C})(\overline{AB\bar{C}})}$$

(teorema di **De Morgan** e doppia negazione)

Per la definizione di / si ha:

$$F = (\bar{A}/B/C)(\bar{A}/\bar{B}/C)(A/B/\bar{C})$$

Quindi: conoscendo la prima forma canonica, si ottiene una espressione della funzione mediante l'operatore di /, sostituendo semplicemente tutti i segni operativi (+), (*) con / e racchiudendo tra parentesi i gruppi di lettere corrispondenti ai prodotti fondamentali.

b) Consideriamo ora la seconda forma canonica:

$$F = (A + B + C)(A + \bar{B} + C)(\bar{A} + B + C)(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + \bar{C})$$

possiamo scrivere:

$$F = (\overline{\bar{A} \cdot \bar{B} \cdot \bar{C}})(\overline{\bar{A} \cdot B \cdot \bar{C}})(\overline{A \cdot \bar{B} \cdot \bar{C}})(\overline{A \cdot \bar{B} \cdot C})(\overline{A \cdot B \cdot C})$$

$$F = (\bar{A} \cdot \bar{B} \cdot \bar{C}) \downarrow (\bar{A} \cdot B \cdot \mathbb{C}) \downarrow (A \cdot \bar{B} \cdot \bar{C}) \downarrow (A \cdot \bar{B} \cdot C) \downarrow (A \cdot B \cdot C)$$

$$F = (A \downarrow B \downarrow C) \downarrow (A \downarrow \bar{B} \downarrow C) \downarrow (\bar{A} \downarrow B \downarrow C) \downarrow (\bar{A} \downarrow B \downarrow \bar{C}) \downarrow (\bar{A} \downarrow \bar{B} \downarrow \bar{C})$$

Donde la regola:

a partire dalla seconda forma canonica, si ottiene una espressione della funzione mediante l'operatore $\downarrow$ sostituendo $(+)$, $(\cdot)$ con $(\downarrow\ )$ e racchiudendo in parentesi i gruppi corrispondenti alle somme fondamentali della seconda forma canonica.

7.8 Funzione OR esclusivo $\oplus$ (XOR)

Anche questa funzione l'abbiamo incontrata già, e sommariamente trattata al capitolo **5.3**.

Oltre che con il nome di *OR* esclusivo, questa funzione la si può incontrare anche sotto il nome di **inequivalenza**, **dilemma** e **addizione** modulo 2. Mentre i primi tre nomi traggono origine dall'algebra delle proposizioni, **addizione modulo 2** trova la sua giustificazione nel fatto che, se si considera come risultato di una somma la classe **resto mod.2** a cui appartiene il risultato vero, allora si ottengono gli stessi valori della funzione **XOR**.

8

Rappresentazione geometrica delle funzioni booleane e loro minimizzazione

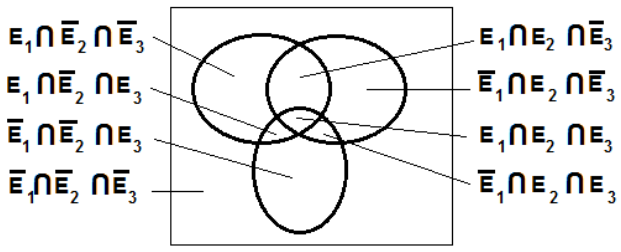
8.1 Diagrammi di Venn

I diagrammi di Venn sono una rappresentazione grafica delle relazioni tra insiemi, e in particolare delle operazioni di unione, intersezione e complementazione. Si è visto inoltre che ogni algebra di insiemi con le operazioni sopra citate è un'algebra di Boole. Ne segue che questi diagrammi di Venn possono essere utilizzati per rappresentare le operazioni sulle funzioni di commutazione che formano esse stesse un'algebra di Boole. Per fare ciò è sufficiente utilizzare la seguente corrispondenza tra gli insiemi e le funzioni booleane: ad ogni insieme E_i si associa la funzione caratteristica X_i.

Questo comporta la corrispondenza tra operazioni su insiemi e funzioni caratteristiche:

$$E_i \cup E_j \iff X_i + X_j$$
$$E_i \cap E_j \iff X_i \cdot X_j$$
$$\bar{E} \iff \bar{X}_i$$

Dati n insiemi $E_1 ... E_n$ rappresentati su un diagramma di Venn, essi definiscono 2^n sottoinsiemi caratterizzati dalla loro **inclusione** o **non inclusione** in ciascuno degli insiemi $E_1 ... E_n$.

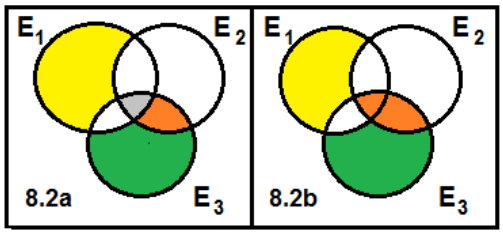


Ogni insieme esistente nel diagramma di Venn può essere definito mediante una selezione degli insiemi $E_1 ... E_n$:

La funzione caratteristica E definita nella tavola (8.1) è quindi rappresentata dall'insieme E dal diagramma di Wenn (8.2).

E è dato dagli insiemi colorati.

Z_n	T_1	Z_n+1
0	0	0
0	1	1
1	0	1
1	1	0



L'insieme E può essere anche definito mediante una espressione:

$$E = (\bar{E}_1 \cap \bar{E}_2 \cap E_3) \cup (\bar{E}_1 \cap E_2 \cap E_3) \cup (E_1 \cap \bar{E}_2 \cap \bar{E}_3) \cup (E_1 \cap E_2 \cap E_3)$$

se ne deduce per X l'espressione:

$$X = \bar{X}_1\bar{X}_2X_3 + \bar{X}_1X_2X_3 + X_1\bar{X}_2\bar{X}_3 + X_1X_2X_3$$

Ci proponiamo di semplificare le espressioni di E e di X

$$\begin{aligned}
E &= (\bar{E}_1 \cap \bar{E}_2 \cap E_3) \cup (\bar{E}_1 \cap E_2 \cap E_3) \cup (E_1 \cap \bar{E}_2 \cap \bar{E}_3) \cup (E_1 \cap E_2 \cap E_3) = \\
&= (\bar{E}_1 \cap \bar{E}_2 \cap E_3) \cup (\bar{E}_1 \cap E_2 \cap E_3) \cup (E_1 \cap \bar{E}_2 \cap \bar{E}_3) \cup (\bar{E}_1 \cap E_2 \cap E_3) \\
&\cup (E_1 \cap E_2 \cap E_3) = \\
&= [(E_1 \cap \bar{E}_1) \cup (E_2 \cap E_3)] \cup [\bar{E}_1 \cap (E_2 \cup \bar{E}_2) \cap E_3] \cup (E_1 \cap \bar{E}_2 \cap \bar{E}_3) = \\
&= (E_2 \cap E_3) \cup (\bar{E}_1 \cap E_3) \cup (E_1 \cap \bar{E}_2 \cap \bar{E}_3) \\
X &= \bar{X}_1\bar{X}_2X_3 + \bar{X}_1X_2X_3 + X_1\bar{X}_2\bar{X}_3 + X_1X_2X_3 \\
&= (X_1X_2X_3 + \bar{X}_1X_2X_3) + (\bar{X}_1\bar{X}_2X_3 + \bar{X}_1X_2X_3) + X_1\bar{X}_2\bar{X}_3 \\
&= [(X_1 + \bar{X}_1)X_2X_3] + [\bar{X}_1(X_2 + \bar{X}_2)X_3]X_1\bar{X}_2\bar{X}_3 \\
&= X_2X_3 + \bar{X}_1X_3 + X_1\bar{X}_2\bar{X}_3
\end{aligned}$$

Ma la semplificazione si può ottenere anche osservando che l'insieme E può essere ottenuto come:

1. riunione di 4 insiemi disgiunti (8.2a)
2. riunione di 3 insiemi disgiunti (8.2b)

Considerando quindi semplicemente il diagramma di Wenn si ottiene:

$$E = (E_1 \cap \bar{E}_2 \cap \bar{E}_3) \cup (E_2 \cap E_3) \cup (\bar{E}_1 \cap E_3)$$

e di conseguenza

$$X = X_1\bar{X}_2\bar{X}_3 + X_2X_3 + \bar{X}_1X_3$$

espressioni ottenute in precedenza in via algebrica.

Riassumendo, il digramma di **Wenn** permette di sostituire le operazioni algebriche con l'esame delle configurazioni geometriche, e questo metodo può essere utilizzato soprattutto per la semplificazione delle espressioni.

8.2 Diagrammi di Karnaugh

Esporremo ora il metodo per la semplificazione delle funzioni dovuto a Karnaugh. Tale metodo ci permetterà di eseguire le minimizzazioni con una certa speditezza, ed è conveniente applicarlo a funzioni con al massimo 6 variabili.

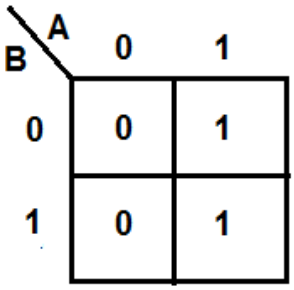
Per poter applicare questo procedimento, è necessario conoscere le rappresentazioni delle funzioni col metodo di Karnaugh. Rappresentazioni che permettono di visualizzare una funzione sotto forma di superficie e sono una applicazione della teoria degli insiemi-Nelle tavole della verità abbiamo visto che, nelle colonne delle variabili, ad ogni riga corrisponde un valore possibile della variabile presa in considerazione.

Nei diagrammi di Karnaugh, anziché far corrispondere una riga, per ogni valore della variabile, si fa corrispondere una superficie delimitata da un quadretto. Costruiamo per esempio il diagramma di Karnaugh per una funzione X dipendente da due variabili:

Tabella 8.1: Tabella A

A	B	$A \cdot B$	$A + A \cdot B$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

$$X = A + A \cdot B$$



La tavola (8.2) della verità di questa funzione ha quattro righe, corrispondenti alle 4 possibili combinazioni dei valori assunti dalle variabili.

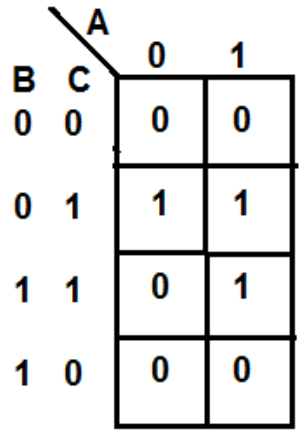
Il diagramma di Karnaugh corrispondente avrà due righe e 4 caselle, essendo i valori di A (prima variabile) posti sopra le caselle orizzontalmente ed i valori della seconda variabile posti a fianco in verticale.

Nei quadretti interni distribuiremo i valori assunti dalla funzione. La tavola della verità (8.2) avrà il diagramma affiancato.

8.2.1 I diagrammi di Karnaugh per tre, quattro o più variabili

La tavola della verità della funzione $X = (A + \bar{B})C$ a tre variabili ha 8 righe.

A	B	C	$A + \bar{B}$	$(A + \bar{B})C$
0	0	0	1	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	1	0	1	1
1	1	0	1	0
1	1	1	1	1



Il corrispondente diagramma di Karnaugh avrà 8 caselle, ognuna associata ad una delle 8 combinazioni possibili, poste su due colonne e 4 righe. A una variabile si assegnano le due colonne, una per il valore 0 l'altra per il valore 1; alle altre variabili si fanno corrispondere le 4 righe, ognuna relativa alle combinazioni:

0 0, 0 1, 1 1, 1 0

Notiamo che nella tavola della verità le combinazioni sono:

0 0, 0 1, 1 0, 1 1

mentre nei diagrammi di Karnaugh avremo le ultime due cifre invertite (vedere tabella 8.4).

Poiché a ogni casella corrisponde il valore che la funzione assume per i particolari valori delle variabili, è stato scelto opportunamente l'ordine delle righe per fare in

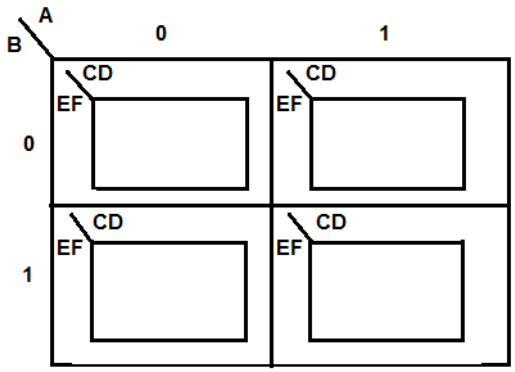


Fig. 8.1: Diagramma di Karnaugh a sei variabili

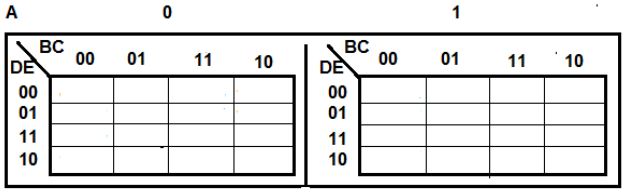
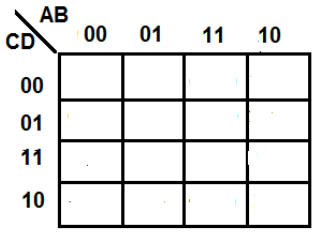


Fig. 8.2: Diagramma di Karnaugh a cinque variabili

modo che, passando da un quadratino al successivo, si abbia il cambiamento di una sola variabile.

8.2.2 Diagramma di Karnaugh a quattro variabili

Una tabella della verità a quattro variabili ha 16 righe corrispondenti a tutte le possibili combinazioni. Il Diagramma di Karnaugh avrà 16 caselle, disposte su quattro righe e quattro colonne. Si assegneranno le quattro colonne alle possibili combinazioni delle prime due variabili e le quattro righe alle combinazioni 00, 01, 11, 10 delle altre due.



Se le variabili fossero 5 o 6 si userebbero dei Diagrammi multipli: la forma qui proposta non è l'unica ma quella di uso più corrente.

Parte II

Circuiti logici e calcolatori digitali

9

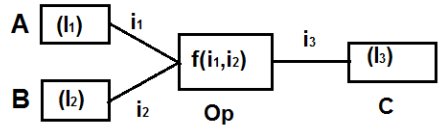
Circuiti logici e di memoria

9.1 Rappresentazione di una informazione digitale

9.1.1 Struttura fondamentale dei circuiti digitali

Si distinguono nei circuiti digitali, tre tipi fondamentali:

1. Circuiti di memorizzazione: cioè capaci di ricevere, conservare, restituire i segnali delle informazioni.
2. Circuiti operatori: cioè capaci di realizzare le funzioni booleane a partire dai segnali delle informazioni.
3. Circuiti di connessione: cioè capaci di trasferire i segnali.



rappresentazione di una informazione digitale

Nella forma più semplice la struttura unitaria è composta di:

- due circuiti di memorizzazione **(A,B)** in cui sono conservate delle informazioni (I_1, I_2) e capaci di emettere i corrispondenti segnali (i_1, i_2).
- un operatore capace di realizzare la funzione dei segnali tali che:

$$i_3 = f(ki_1, i_2) - - - - - - - - - I_3 = f(I_1, I_2$$

- un circuito di memorizzazione **(C)** che riceve il segnale (i_3) emesso dall'operatore e capace di conservare l'informazione corrispondente I_3.
- connessioni tra questi circuiti.

Un segnale I, rappresentativo di una informazione I, sarà una grandezza fisica suscettibile di assumere nel tempo due diversi valori determinati dai circuiti emettitori e operatori.

Si cercano in genere quei dispositivi, legati a grandezze fisiche, capaci di assumere due distinti stati di equilibrio stabile.

Una generica informazione sarà rappresentata da un insieme di segnali elementari. Nei calcolatori elettronici moderni i segnali rappresentativi delle informazioni sono generalmente costituiti da tensioni o correnti elettriche.

:-Segnali elettrici rappresentativi delle informazioni.

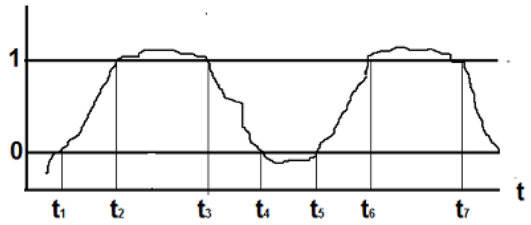
Precisiamo il senso booleano che si può attribuire ad un segnale trasmesso nell'insieme dei componenti di un calcolatore digitale. Consideriamo un segnale elettrico variabile nel tempo (ad esempio, una tensione): $A = V(t)$.

Consideriamo due livelli di riferimento m_0 e m_1 con la seguente convenzione sulla informazione binaria contenuta in $V(t)$:

$$\begin{array}{ll} 1 & V(t) \leq m_1 \\ 0 & V(t) \leq m_0 \end{array}$$

Naturalmente le cifre binarie 0 ed 1 non implicano che l'informazione sia espressa nella **base 2**, ma solo che l'informazione è espressa in un linguaggio a due distinti valori che potrebbero essere rappresentati dagli elementi di una generica coppia $(+, -)$, $(_0, -)$.

La differenza Δ fra i due livelli è determinata dal potere risolutivo degli strumenti e determina a sua volta la differenza fra massima e minima ampiezza dei possibili segnali.



Considerando la figura 1.2 notiamo che, nel tempo, il segnale è:
perfettamente definito in uno dei due possibili valori negli intervalli

$$(t_2, t_3), (t_6, t_7), (t_4, t_5, t_8), (t_7), ...$$

indeterminato negli intervalli

$$(t_1, t_2), (t_3, t_4), (t_5, t_6), (t_7, t_8), ...$$

e di conseguenza il segnale deve essere utilizzato solo in istanti ben determinati dagli intervalli di ambiguità.

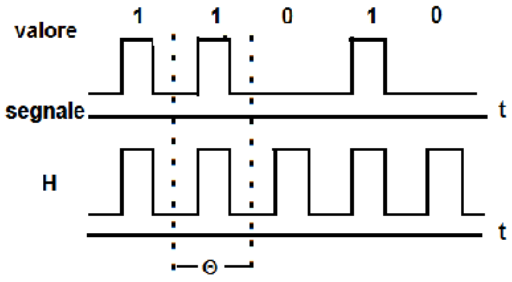
9.1.2 Strutture sequenziali

L'analisi tecnica dell'andamento temporale del segnale, in funzione dei componenti del circuito e nelle condizioni più sfavorevoli, corrispondenti a zone di ambiguità di maggiore larghezza, permette di stabilire gli intervalli temporali entro i quali ha senso esaminare il valore dell'informazione.

Il succedersi, nel tempo, di questi intervalli, può essere regolare o irregolare. Nel secondo caso, si parla di circuiti asincroni ed è il funzionamento interno di ogni circuito che determina gli intervalli opportuni.

Più frequentemente si ha una ripetizione regolare nel tempo di tali intervalli (circuiti sincroni) con periodo Θ. Si può dire che, durante l'intervallo di tempo di un periodo binario, il segnale è atto a rappresentare una informazione binaria elementare. Oggi si superano valori di periodo corrispondenti ai 10 nano-secondi (=!0^-9 sec).

Consideriamo adesso una informazione rappresentata da impulsi secondo la convenzione:



- valore 1: presenza di impulso
- valore 0: assenza di impulso

cioè avendo sostituito alla nozione di livello rappresentativo quella di presenza o assenza di impulso.

Nella figura si è riportato un segnale ideale di impulso, cioè privo di zone di ambiguità, per mettere in evidenza che esistono altre cause per le quali il segnale è significativo solo in brevi intervalli di tempo, e cioè si deve escludere anche il tempo che il segnale impiega per ripristinare la condizione iniziale (eventuale ritorno a zero).

Gli intervalli di tempo significativi sono scanditi con un segnale di riferimento H, detto segnale di orologio. Si può aggiungere che il segnale di orologio è rappresentativo della tautologia.

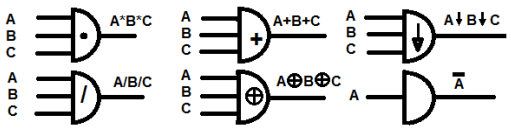
9.1.3 Convenzioni della logica

			livello positivo			livello negativo		
X	Y	F	X	Y	F	X	Y	F
-	-	-	0	0	0	1	1	1
-	+	-	0	1	0	1	0	1
+	-	-	1	0	0	0	1	1
+	+	+	1	1	1	0	0	0
					$F = XY$			$F = X + Y$

La scelta che, nel caso appena visto, ha associato il valore 1 alla presenza di impulso è arbitraria. Distinguiamo una logica (intesa qui in pratica come metodo di realizzazione elettronica di operatori logici) positiva, relativa alla convenzione di associare il valore 1 al livello più alto (+) ed una logica negativa alla convenzione il valore 0al livello inferiore (-).

In generale, se con una convenzione si ha funzione booleana, con la convenzione opposta si ha la funzione duale, infatti:

$$\bar{F} = (\bar{x}_1, \bar{x}_2, ..., \bar{x}_n) =\sim F = (x_1, x_2, ..., x_n)$$

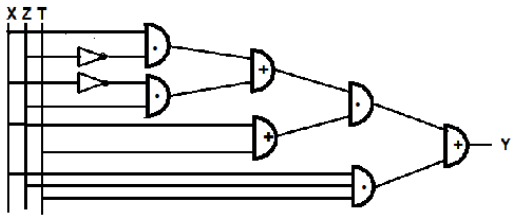


9.1.4 Porte e livelli

Ai circuiti che realizzano i vari operatori logici si dà il nome di porte. Nella figura si hanno il simbolo in standard DIN (acronimo di Deutsches Institut fuer Normun) di complementazione e quelli, pure in standard DIN, di AND, OR, XOR, NOR, $NAND$ a tre ingressi.

Si dà il nome di livelli ai successivi stadi determinati dagli operatori elementari, attraverso i quali passa il segnale prima di giungere alla realizzazione del segnale risultante.

Considerando le funzioni booleane si vede che ad ogni livello si ha un cambiamento di forma (disgiuntiva o congiuntiva).



Esempio: consideriamo la funzione

$$Y = (X\bar{Z} + \bar{X}Z)(X + T) + (X\ Z\ T)$$

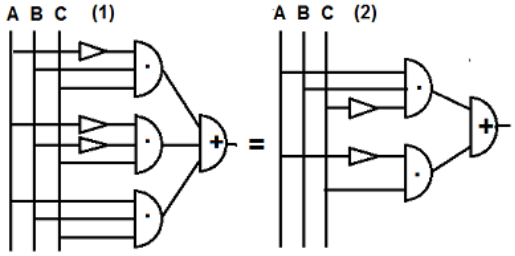
Si hanno 4 livelli e 7 porte:

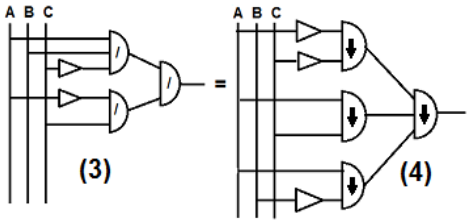
La nozione di livello è molto importante dal punto di vista tecnico, essendo il numero dei livelli pari al numero massimo di porte attraversate successivamente da un segnale fornisce una stima del tempo di propagazione del segnale.

Notiamo infine che non tutti i segnali devono necessariamente percorrere lo stesso numero di livelli. In una tecnica sincrona è necessario per tenerne conto compensare le differenze con opportuni ritardi.

Ogni forma booleana elementare richiede, per essere realizzata, almeno due livelli di operatori elementari.

9.1.5 Circuiti combinatori e circuiti sequenziali





Un circuito di commutazione è un circuito i cui segnali seguono una logica a due valori, come nel caso booleano. Distinguiamo due tipi di circuiti: combinatori e sequenziali.

Un circuito combinatorio è un circuito tale che ad ogni insieme di stati d'ingresso completamente specificato fa corrispondere un ben determinato insieme di stati d'uscita. Ad uno stesso insieme di stati in uscita possono corrispondere più insiemi di stati in ingresso.

Ad esempio, i circuiti in figura (1), (2), (3) e (4):

$$\bar{A}BC+\bar{A}\bar{B}C+AB\bar{C} = AB\bar{C}+\bar{A}C = (A/B/\bar{C})/(\bar{A}/C) = (\bar{A}\downarrow\bar{C})\downarrow(A\downarrow C)\downarrow(A\downarrow\bar{B})$$

Un circuito sequenziale è un circuito per il quale delle variabili interne, associate ad elementi di memoria, intervengono nella funzione di uscita. L'insieme degli stati di uscita è quindi una funzione non solo delle variabili di ingresso, ma anche degli stati precedenti.

9.2 Circuiti di memoria

9.2.1 Introduzione del tempo nelle funzioni booleane

La nozione del tempo è indissociabile dalla formulazione delle funzioni in dipendenza dalla evoluzione temporale delle operazioni.

Notiamo che l'algebra di Boole non permette di introdurre il tempo come variabile indipendente nella formulazione delle funzioni.

Ciò costituisce un inconveniente soprattutto nello studio dei circuiti sequenziali; per lo più si cerca di ricondurre lo schema di un circuito sequenziale a quello di un circuito combinatorio in cui gli insiemi di stato delle variabili sono legati da un ordine logico dipendente dal tempo.

Per lo studio dei circuiti digitali si opera una discretizzazione del tempo in intervalli (regolari o non). L'insieme di questi intervalli costituisce la sequenza temporale fondamentale:

$$\tau = \left\{...t_{-3}, t_{-2}, t_{-1}, t_0, t_1, t_2, t_3, ...\right\}$$

$$t_0$$

è l'istante attuale

$$...t_{-3}, tg_{-2}, t_{-1}$$

sono gli istanti precedenti

$$t_1, t_2, t_3...$$

sono gli istanti successivi

9.2.2 Ritardo

Come prima conseguenza dell'introduzione del tempo, bisogna determinare dei circuiti atti a restituire una informazione in un intervallo di tempo diverso da quello della sua formazione.

Un primo procedimento di spostamento nel tempo di una informazione consiste nel fare percorrere al segnale rappresentativo, un elemento di ritardo tale che il segnale risultante sia sempre rappresentativo della stessa informazione:

Esprimiamo la funzione di ritardo tramite un operatore Δ che agisce sulla sequenza temporale:

A_n	$(A_n)\Delta_x = Y_{n+x}$
0	0
1	1

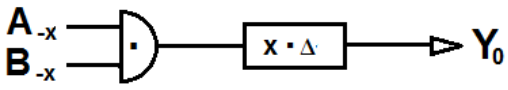
(essendo X il ritardo scritto in numero di periodi Θ della sequenza temporale) si ha:

$$Y_{n+x} = \Delta_x(A_n) = A_n$$

L'operatore Δ è commutativo e associativo.

9.2.3 Funzioni sequenziali elementari

Combinando l'operatore Δ con una generica operazione booleana si realizza una funzione sequenziale elementare, cioè una funzione booleana in cui il tempo interviene come variabile indipendente.



A_{-x}	B_{-x}	Y_0
0	0	0
0	1	0
1	0	0
1	1	1

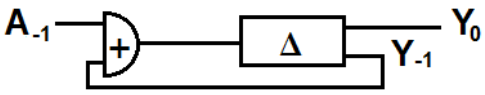
Ad esempio:

$$Y_0 = \Delta_x(A_{-x} \cdot B_{-x})$$

9.2.4 Circuiti chiusi (o ad anello)

La combinazione dell'operatore Δ e di operatori booleani conducono alla realizzazione di particolari funzioni sequenziali, in cui una delle variabili di ingresso è la funzione di uscita. La realizzazione pratica consiste in un circuito comprendente un percorso chiuso o anello, in cui una connessione riporta fra gli ingressi un valore di uscita (feedback).

Esempio 1:



Y_{-1}	A_{-1}	Y_0
0	0	0
0	1	1
1	0	1
1	1	1

$$Y_0 = \Delta_1(A_1 + Y_1)$$

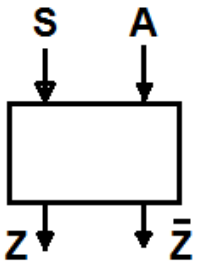
$$(\tau = 1 \cdot \delta)$$

Questo circuito funziona da memoria.

Esempio 2:

Questo circuiti fornisce risposta affermativa in funzione della disparità della sequenza binaria A.

9.2.5 Memorie elementari



Un secondo procedimento (cfr. il ritardo) di spostamento nel tempo di una informazione booleana consiste nel registrarne opportunamente il valore conservandolo in memoria.

Nel caso delle informazioni binarie elementari, questa funzione è svolta da un operatore logico Γ detto memoria elementare.

$$\Gamma(S, R) = Z$$

I due segnali esterni, S ed R, impongono all'uscita Z i valori:

$$S = 1 \qquad Z = 1 \qquad \bar{Z} = 0$$
$$R = 1 \qquad Z = 0 \qquad \bar{Z} = 1$$

Questo elemento di memoria registra il valore di un segnale S e conserva un segnale rappresentativo di tale valore finché un segnale R non lo riporta a zero.

L'operatore $\Gamma(S, R)$ quindi è suscettibile di:

1. ricevere un segnale
2. conservare il segnale ricevuto
3. essere azzerato

Si suddividono i circuiti di memoria in due categorie in funzione delle strutture tecniche richieste:

a memorie elementari statiche: la cui realizzazione tecnica più usuale è il flip-flop (ved,§ 1.3.3).

b memorie dinamiche elementari: sono basate sulla presenza o assenza di circolazione di un impulso in un circuito di ritardo. Quindi, al contrario per esempio della linea di ritardo costituita da elementi bipolari, una memoria dinamica necessita di un circuito chiuso o ad anello.

Flip-flop S-R

Dal punto di vista funzionale il **F/F S-R** presenta due ingressi detti S (Set) e R (Reset), e due uscite complementari Z e $\bar{Z}$. Il funzionamento è il seguente:

- un segnale su S mette o lascia il **F/F** nello stato 1 (iscrizione dell'1)
- un segnale su R mette o lascia il **F/F** nello stato 0 (iscrizione dell'0)
- un segnale nullo lascia invariato il **F/F**
- la combinazione **11** in ingresso è proibita

Lo stato interno della memoria è accessibile mediante l'esame della combinazione $Z\bar{Z}$ di uscita fra le due possibili (01 e 10). Si può introdurre anche una variabile di stato interno Q e si hanno le due possibili scelte:

$$Q = Z \qquad \textit{oppure} \qquad Q = \bar{Z}$$

Assumendo $Q = Z$ il funzionamento di un **F/F S-R** è rappresentato nella tabella seguente:

n	Z_n	S_n	R_n	$Z_n + 1$	ingressi:	stato interno:
0	0	0	0	0	nulla	invariato
1	0	0	1	0	iscriz. di 0	invariato
2	0	1	0	1	iscriz. di 1	$0 \to 1$
3	0	1	1	-	contraddittori	incerto
4	1	0	0	1	nulla	invariato
5	1	0	1	0	iscriz. di 0	$1 \to 0$
6	1	1	0	1	iscriz. di 1	invariato
7	1	1	1	-	contraddittori	incerto

Risulta:

$$Z_{n+1} = \bar{Z}_n S_n \bar{R}_n + Z_n \bar{S}_n \bar{R}_n + Z_n S_n \bar{R}_n = (S_n + Z_n)\bar{R}_n$$

Se inoltre la combinazione logica $S_n R_n = 1$ (corrispondente alla combinazione 3 e 7 della tabella) è resa materialmente impossibile si può fare uso dei termini eliminati $\bar{Z}_n S_n R_n$ e $Z_n S_n R_n$ per minimizzare ulteriormente l'espressione, ottenendo:

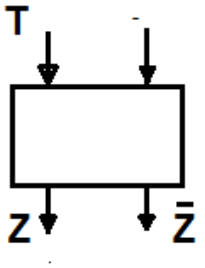
$$Z_{n+1} = S_n + Z_n \bar{R}_n$$

Flip-flop J-K

Si comporta come un **F/F S-R** con J ingresso di **set**. È però ammessa la combinazione **11** in ingresso e l'effetto è quello di far cambiare lo stato interno. Risulta quindi:

$$Z_{n+1} = \bar{Z}_n S_n \bar{R}_n + Z_n \bar{S}_n \bar{R}_n + Z_n S_n \bar{R}_n + \bar{Z}_n S_n R_n = \bar{Z}_n S_n + Z_n \bar{R}_n$$

Flip-flop T



Z_n	T_1	$Z_n + 1$
0	0	0
0	1	1
1	0	1
1	1	0

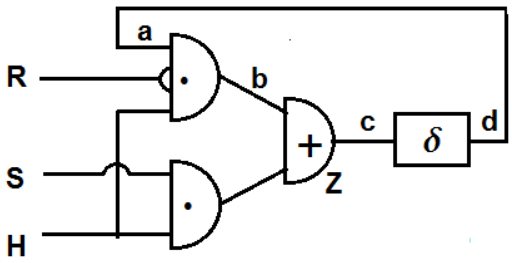
È un flip-flop a un solo ingresso T e tale da cambiare stato ad ogni segnale in ingresso.

$$Z_{n+1} = T_n \bigoplus Z_n$$

9.2.6 Memorie elementari dinamiche

Flip-flop dinamico di tipo S-R

Consideriamo il circuito in figura . I segnali d'ingresso arrivano in S e in R, l'ingresso H è quello dell'orologio: il segnale applicate consiste in una successione regolare di impulsi. In uscita della porta OR si trova un ritardo δ pari al periodo dell'orologio.



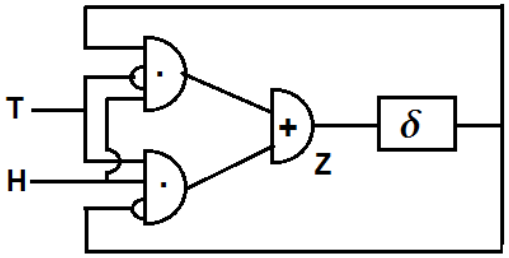
In tal modo il circuito funzione come un flip-flop a due stati caratterizzati dalla presenza o dall'assenza di una ricircolazione di un impulso nel percorso chiuso **abcda**.

Consideriamo per esempio uno stato iniziale caratterizzato dall'assenza di impulso in **abcda**.

Un impulso su S in sincronismo con H ricircola con periodo δ finché non arriva un impulso in R(con cui si blocca il rientro dell'impulso) e questo sia che si applichi o no degli impulsi in S. Viceversa, se si applica un impulso su R in sincronismo con H, l'impulso circolante che si trova in a non può rientrare nell'anello e la ricircolazione è interrotta. Si è così ritornati allo stato iniziale. Questo è il funzionamento del **F/F** di tipo **S-R** descritto prima, a parte il fatto che è permessa la combinazione di ingresso $S = 1$, $R = 1$ per $Z_n = 0$, 1 con $Z_{n+1} = 1$. Ciò permette di utilizzare la seconda espressione minimizzata trovata per il **F/F S-R** e, aggiungendo la variabile H_n rappresentativa dell'orologio si verifica che il circuito fornisce la:

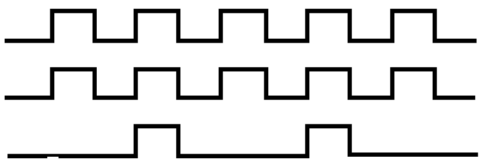
$$z_{n+1} = H_n(S_n + Z_n\bar{R}_n)$$

Flip-flop dinamico tipo T



Con la medesima convenzione fatta per il flip-flop SR si ha, per il circuito a lato,

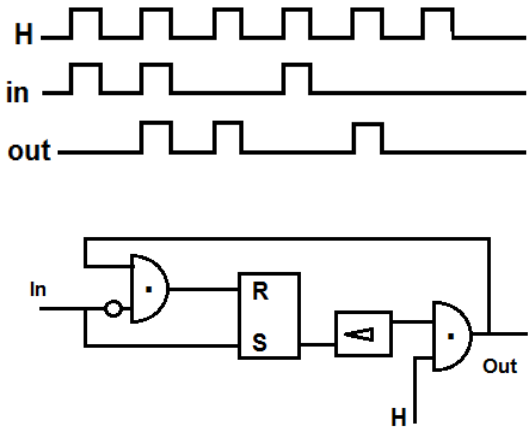
$$Z_{n+1} = H_n(T_n \oplus Z_n)$$



che realizza un flip-flop di tipo T. L'uscita Z, quando si applica un impulso in T, vale 1 ogni due impulsi, realizza quindi un divisore per due:

sequenza d'ingresso e di uscita per un flip-flop di tipo T con stato interno inizialmente nullo.

Circuito dinamico di ritardo



Il circuito in figura fornisce in uscita una sequenza uguale a quella in ingresso, ma ritardata di un periodo di orologio.

Un impulso 1 in ingresso pone in 1 il flip-flop: l'impulso in uscita viene ritardato di un periodo e passa, in sincronia con l'impulso di orologio, per la porta di "AND". L'uscita è l'impulso ritardato. Questo impulso di uscita viene rinviato tramite una porta di "AND" all'ingresso "R" solo se il nuovo impulso è zero. Se invece il nuovo impulso fosse stati "1" l'impulso di reset sarebbe stato inibito dalla relativa porta di "AND". Si può verificare che il circuito, con $R_n = \bar{S}_n Z_n$, fornisce la:

$$Z_{n+1} = H_n(\bar{Z}_n S_n + Z_n S_n) = H_n S_n$$

9.3 Aspetti tecnologici

9.3.1 Tecniche a semiconduttori

La comparsa sul mercato di singoli componenti semi-conduttori ha rivoluzionato la teoria dei circuiti elettronici.

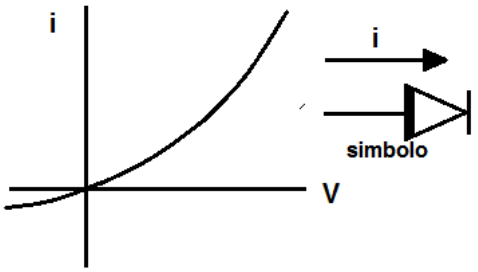
I semi conduttori sono conosciuti da tempo (primi rivelatori radio), ma solo dopo il 1950 la loro realizzazione industriale ha ottenuto i requisiti necessari per poter sostituire i tubi a vuoto.

Un semiconduttore "intrinseco" cioè allo stato puro è un conduttore cattivo in una vasta zona di temperatura. L'introduzione di impurità modifica profondamente le proprietà di conducibilità elettrica: sin hanno allora i semi-conduttori estrinseci. A seconda della natura della impurità il materiale semi-conduttore si comporta come donatore o come accettore di elettroni. Nel primo caso si dice di tipo N (negativo) e la conducibilità avviene per trasporto di elettroni, nel secondo caso si dice di tipo P (positivo) e la conducibilità viene descritta in termini di **buche** o "lacune positive"

Uno dei semi-conduttori più usati è il cristallo di germanio (Ge). I cristalli semi conduttori sono ottenuti, a partire dal cristallo puro, per introduzione di impurità: antimonio (Sb), bismuto (Bi) o arsenico (As) per il tipo N; alluminio (Al), boro (B), rame (Cu) per il tipo P. Oltre il germanio sono in uso anche il selenio ed il silicio.

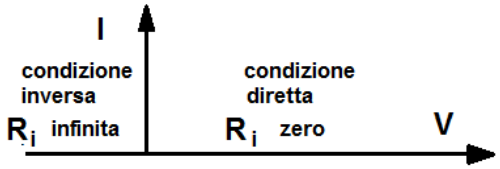
Diodi a giunzione

Sono realizzati dal contatto di cristalli di tipo P e N: si forma una barriera di potenziale che contrasta il passaggio della corrente elettrica in una delle due direzioni.



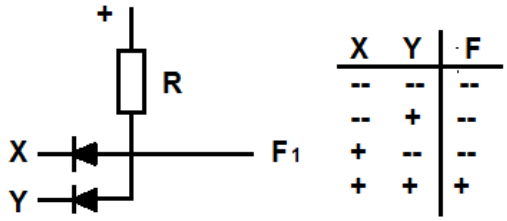
La curva caratteristica i in funzione della tensione V ai capi è riportata nel grafico a latere.

Da quanto accennato sui diodi a proposito dei tubi a vuoto e dei semi-conduttori si può astrarre il comportamento del diodo ideale: elemento unidirezionale capace di far scorrere in un solo senso una corrente comunque elevata con resistenza interna R_i e differenza di tensione ai capi nulla; mentre presenta infinita resistenza interna al passaggio di corrente nel senso opposto. Il comportamento dipende dal segno della differenza di potenziale applicata (polarizzazione diretta e inversa, rispettivamente).



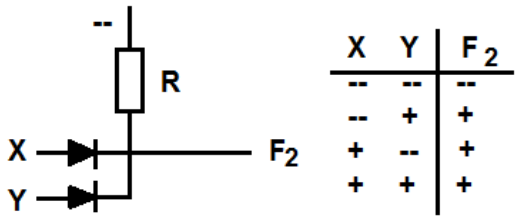
Sulla destra grafico del comportamento di un diodo ideale

Porte a diodi



La figura di fianco rappresenta una porta a diodi e resistori. Schematizzando i diodi come ideali si vede che se una o entrambe le tensioni di ingresso sono negative (-), inferiori alla tensione +, l'uscita avrà tensione −, viceversa se entrambi gli ingressi hanno tensione + l'uscita avrà tensione +. Nel primo caso si ha passaggio di corrente e nel secondo no. Con i diodi reali si avrà valore non nullo della resistenza nel primo caso e valore finito nel secondo.

Analogamente dalla figura qui affiancata segue la costruzione della funzione "F2".

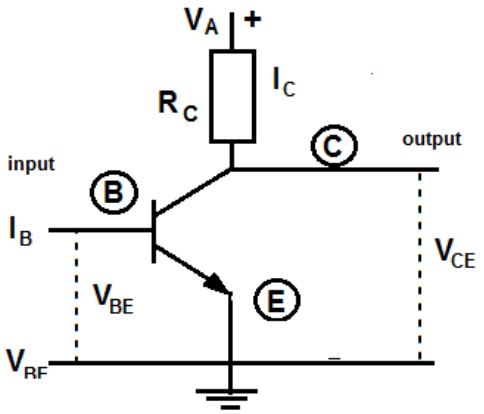


In logica positiva (conf.par,1.1.3) il circuito attinente funzione **F1** realizza una porta AND, mentre quello attinente funzione **F2** realizza una porta OR.

Con la convenzione logica negativa i due circuiti realizzano le funzioni duali e cioè OR e AND' rispettivamente.

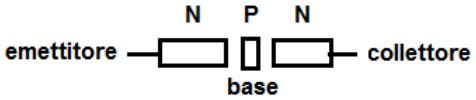
L'inconveniente maggiore nell'uso del semi-conduttori risiede nelle condizioni di funzionamento. Nonostante i continui miglioramenti è sempre necessario, con precauzioni di condizionamento di aria, di refrigerazione... mantenere il funzionamento in un ristretto intervallo di temperatura

9.3.2 Transistori



Notiamo che le porte AND e OR non sono sufficienti per la realizzazione di una funzione booleana; da un punto di vista logico è necessario aggiungere un circuito atto a realizzare la complementazione.

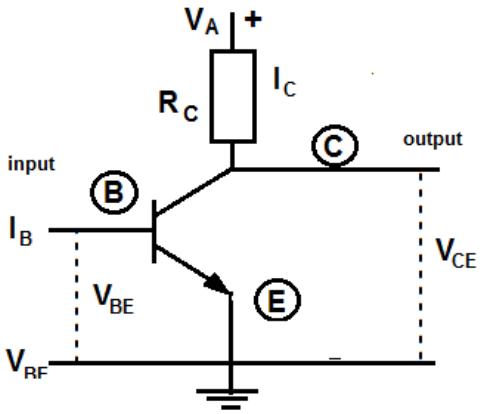
Questo tipo di circuito non si può ottenere con i componenti visti sinora, che erano tutti di tipo passivo. Infatti la complementazione di un segnale può richiedere che il segnale passi da un livello basso ad uno alto, cioè debba essere amplificato, e questo può essere ottenuto solo con componenti attivi. I componenti attivi sono necessari anche per la rigenerazione di un segnale che, a causa dell'attenuazione, rischi di essere inferiore del suo livello rappresentativo.



Fra gli elementi attivi facciamo cenno ai transistori. Il transistor a giunzione comprende due giunzioni che separano tre regioni semi-conduttrici di cui le estreme

dello stesso tipo e quella intermedia di tipo opposto. Si hanno quindi le due forme possibili **PNP** e **NPN**.

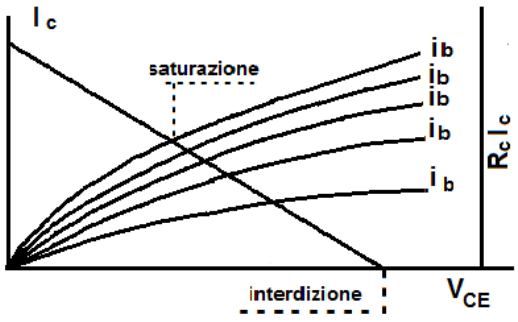
Una delle due estremità, l'emettitore, molto ricco di impurità, può emettere un gran numero di cariche che sono più o meno raccolte dall'altra estremità, il collettore, formato di cristalli meno ricchi di impurità.



Il flusso di cariche è regolato dallo stato della regione intermedia, la base. Poiché devono rappresentare gli stati 0 e 1 in modo che risultino distinguibili senza ambiguità, i transistori vengono fatti lavorare esclusivamente nelle due opposte condizioni di "interdizione" e "saturazione"

Schematizziamo il comportamento ideale di un transistor: mandando alla base una corrente nulla, è nulla la corrente di collettore; il transistor si comporta come un contatto aperto (interdizione) e la tensione di collettore è uguale a quella di alimentazione, non essendo il resistore R percorso da corrente. Inversamente mandando alla base una debole corrente il transistor si comporta come un corto circuito (saturazione), la tensione di collettore va a zero e quindi ai capi del resistore R si ha una differenza di potenziale pari alla tensione di alimentazione.

Il comportamento reale di un transistore è rappresentato dalle seguenti caratteristiche:



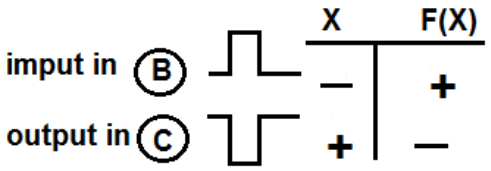
Quindi con una debole corrente di base (dell'ordine delle decine di μA) si regola una corrente di collettore da valori molto bassi a valori dell'ordine di grandezza dei mA; e quindi, essendo la corrente di collettore maggiore di quella necessaria per controllarla, si parla di **guadagno di corrente**.

Corrispondentemente si ha, in interdizione, una differenza di tensione $(\Delta V)_{int}$ non nulla ai capi del transistor e la tensione del collettore non è quella di alimentazione bensì quella di alimentazione diminuita di $(\Delta V)_{int}$: $(V_{CE})_{int} = V_A - (\Delta V)_{int}$ essendo V_A la tensione di alimentazione.

Nel caso di saturazione si ha ai capi del transistor una differenza di tensione non più pari alla tensione di alimentazione, ma leggermente inferiore, per cui la tensione di collettore assume un valore non nullo $(V_{CE}) = V_A - (\Delta V)_{sat}$.

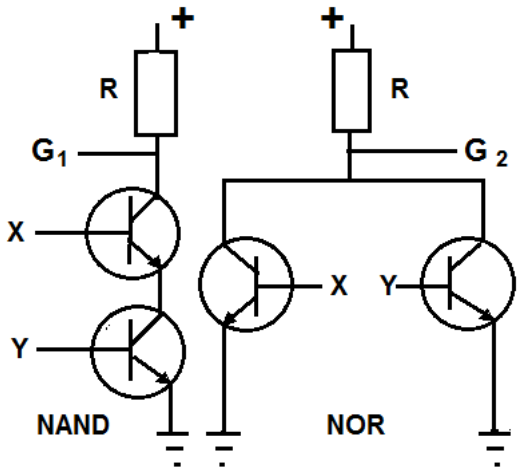
I valori $(V_{CE})_{sat}$ e $(V_{CE})_{int}$ sono tali da essere considerati rappresentativi dei valori logici 0 e 1 in logica positiva.

Invertitore a transistor

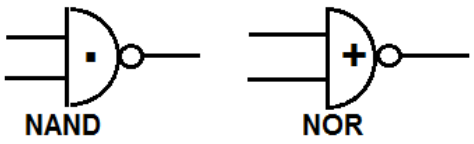


Per quanto detto il transistor tipo **PNC** nel circuito aggregato realizza una funzione **NOT**: infatti lavorando nelle opposte zone di saturazione e interdizione è tale da fornire per un input in B come nel circuito l'output riportato in C, al limite per un transistor ideale con la tabella della verità in cui X è il valore di input e $F(x)$ quello di output.

Circuiti NAND e NOR a transistor



X	Y	G_1	G_2
-	-	+	+
-	+	+	-
+	-	+	-
+	+	-	-



L'interesse dei circuiti che realizzano gli operatori $NAND$ e NOR consiste nel fatto che detti operatori costituiscono, singolarmente, degli insiemi funzionalmente completi (cfr. Parte 1, Cap. VII).

In linea di principio gli elementi visti fin qui sono sufficienti per formare un circuito $NAND$ o NOR. Infatti è sufficiente un circuito formato da una porta a diodi (paragrafo 1.3.2), e da un invertitore a transistor. Il circuito così formato appartiene alla famiglia dei **D-T-L** (Diodo-Transistor-Logic).

I circuiti affiancati fanno parte della famiglia **D-C-T-L** (Direct-Coupled-Transistor-Logic) e realizzano le funzioni $NAND$ e NOR tramite l'uso dei transistor in luogo dei diodi.

Consideriamo il circuito $NAND$: solo se entrambi i transistor conducono ci sarà una caduta di tensione ai capi del resistore ed il potenziale in G_1 sarà basso. Invece considerando il circuito NOR risulta che è sufficiente che uno solo dei due transistor conduca perché il potenziale di G_2 sia basso. Ne segue la tabella della verità mostrata.

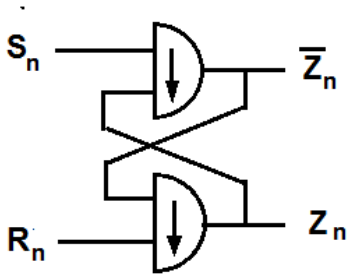
Analogamente a quanto detto a proposito delle porte a diodi la funzione G_1 e G_2 in logica positiva realizzano le funzioni $NAND$ e NOR rispettivamente, mentre in logica negativa realizzano le funzioni duali NOR e $NAND$ rispettivamente.

In simboli i circuiti $NAND$ e NOR oltre che come indicato al paragrafo 1.1.4 possono anche essere indicati con i simboli del codice DIN qui raffigurati.

Flip-flop

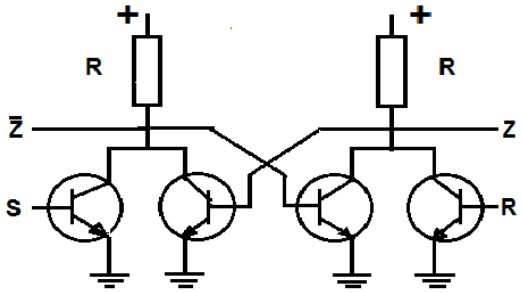
Quanto visto finora è sufficiente per comprendere la struttura di un flip-flop, il cui funzionamento logico si è visto al paragrafo 1.2.4.

Limitiamoci a considerare il caso in cui la combinazione $S_n = R_n = 1$ è proibita, riscriviamo la funzione di uscita $Z_n + 1$ ed il suo complemento $\bar{Z}_{n+1}$ in funzione degli ingressi S_n e R_n tramite l'operatore NOR:



$$Z_{n+1} = (S_n + Z_n) \cdot \bar{R}_n = \overline{(S_n + R_n)} \downarrow R_n = (S_n \downarrow Z_n) \downarrow R_n$$

ed inoltre, considerando che in questo caso si ha $S_n \bar{R}_n = S_n$ si ha anche:



$$\bar{R}_{Z+1} = \overline{(S_n + Z_n) \cdot \mathbb{R}_n} = \overline{S_n + Z_n \cdot \bar{R}_n} = \overline{S_n + (\bar{Z}_n \downarrow R_n)} = S_n \downarrow (\bar{Z}_n \downarrow R_n)$$

Queste due funzioni sono realizzate col circuito logico evidenziato a lato.

È ora sufficiente considerare due circuiti NOR, e connettere simmetricamente l'uscita di uno dei due circuiti ad uno dei due ingressi dell'altro per ottenere uno schema semplificato di flip-flop.

Pertanto, quando la coppia di transistor di ingresso S e Z conduce (caso di saturazione) il potenziale del comune collettore (segnale $\bar{Z}$) mantiene a livello basso quello di una delle basi dell'altra coppia di transistor che, per quanto detto, si trova in condizione di interdizione.

Vi sono pertanto due stati di equilibrio stabile, caratterizzati dal fatto che una delle due coppie di transistor è in saturazione e l'altra in interdizione.

Quando il circuito si trova in uno dei due stati, identificabili tramite le funzioni di uscita Z e $\bar{Z}$, vi rimane indefinitamente, realizzando cioè la funzione di memoria. Tramite segnali di comando esterni si ottiene che il circuito commuti all'altro dei due possibili stati; questi segnali (S e R) vengono mandati a una delle basidi una coppia di transistor. Il passaggio di un flip-flop dallo stato 0 allo stato 1 si dice **setting**, quello dallo stato 1 allo stato 0 di **resettig**. Il comando che provoca la commutazione può essere o un livello di tensione (comando in continua) o impulso di tensione (comando in alternata).

10

Circuiti di un calcolatore digitale (a)

10.1 Addizionatore e sottrattore binario

Siano le variabili x_i, y_i, r_i, S_i e r_{i+1} definite nel capitolo 2, a proposito della somma e della sottrazione di due numeri binari.

Assumendo la convenzione di rappresentate l'i aritmetico mediante l'! della logica booleana, lo 0 aritmetico mediante lo 0 booleano, e utilizzando gli stessi simboli di variabile per i due tipi di grandezza - si possono estrarre dalle tavole del capitolo 2 le tavole di verità che definiscono, per un singolo rango, la somma e la sottrazione.

10.1.1 Semiaddizionatore (half-adder)

Questo circuito è un blocco fondamentale costituito da 3 porte di AND e una porta di OR. Il suo funzionamento viene stabilito mediante la tavola di verità:

X_i	Y_i	S_i	R_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

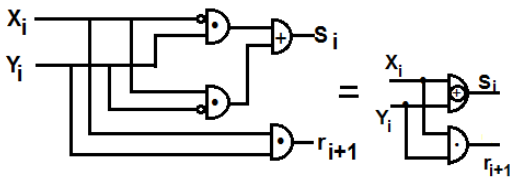
È da notare che questo circuito presenta due output separati: uno per la somma e l'altro per il riporto.

Le due funzioni booleane descritte dalla tavola sono:

$$S_i = Y_i\bar{X}_i + X_i\bar{Y}_i = X_i \oplus Y_i$$

$$r_{i+1} = X_i \cdot Y_i$$

Il diagramma logico di un semiaddizionatore sarà quindi:



Il circuito che genera la funzione S_i viene anche impiegato come porta di $\oplus$ (OR esclusivo)

10.1.2 Addizionatore elementare (Full Adder)

Il processo della somma binaria consiste in questo caso nella addizione contemporanea delle due variabili X_i e Y_i e del riporto r_i ottenuto durante la somma parziale precedente. La tabella della verità relativa S_i e r_{i+1} sarà quindi:

X_i	Y_i	r_{i+r}	S_i	r_{i+r}
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

Equazione booleana di un addizionatore

Forma canonica:

$$S_i = \bar{X}_i\bar{Y}_i r_i + \bar{X}_i Y_i \bar{r}_i + X_i\bar{Y}_i\bar{r}_i\bar{r}_i + X_i Y_i r_i$$

Altre espressioni:
Per S_i, si può scrivere:

$$\begin{aligned} S_i &= (\bar{X}_i\bar{Y}_i + X_iY_i)r_i + (\bar{X}_iY_i + X_i\bar{Y}_i)r_i \\ &= \overline{(X_i \oplus Y_i)} + (X_i \oplus Y_i)\bar{r}_i \\ &= (X_i \oplus Y_i) \oplus r_i \\ &= X_i \oplus Y_i \oplus r_i \end{aligned}$$

La funzione r_{i+1} essere semplificata:

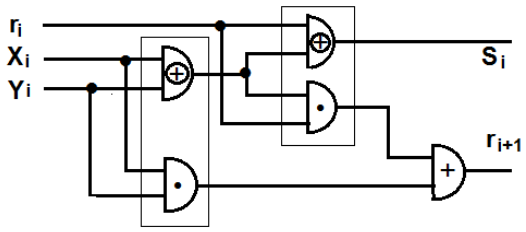
$$\begin{aligned} r_{i+1} &= \bar{X}_iY_ir_i + X_i\bar{Y}_ir_i + X_iY_i\bar{r}_i + x_iY_ir_i + X_iY_ir_i + X_iY_ir_i \\ &= (\bar{X}_i + X_i)Y_ir_i + (\bar{Y}_i + Y_i)X_ir_i + (r_i + \bar{r}_i)X_iY_i \\ &= X_iY_i + Y_ir_i + r_iX_i \\ &= X_iY_i \oplus Y_ir_i \oplus r_iX_i \end{aligned}$$

Per comodità, possiamo scrivere ulteriormente:

$$S_i = (X_i \oplus Y_i) \oplus r_i$$

$$\begin{aligned} r_{i+1} &= X_iY_i(r_i + \bar{r}_i) + r_i(X_i\bar{Y}_i + \bar{X}_iY_i) \\ &= X_iY_i + r_i(X_i \oplus Y_i) \end{aligned}$$

Quindi S_i e r_{i+1} possono essere ottenuti mediante due reti: la prima sarà $X_i \oplus Y_i$ e X_iY_i, la seconda $(X_i \oplus Y_i) \oplus r_i$ e $(X_i \oplus Y_i)r_i$.



Ciascuna di queste reti è un semiaddizionatore.

Connettendole come da figura e utilizzando una porta supplementare OR per formare r_{i+1}, si ottiene un addizionatore elementare a tre ingressi X_i, Y_i, r_i.

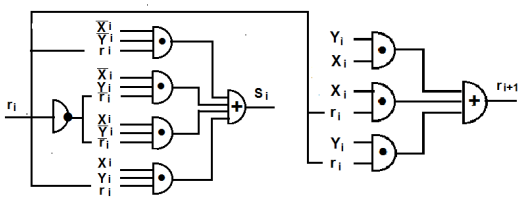
Mediante altre trasformazioni algebriche, si possono ovviamente ottenere altri schemi.

Per esempio, considerando le espressioni:

$$S_i = \bar{X}_i\bar{Y}_ir_i + \bar{X}_iY_i\bar{r}_i + X_i\bar{Y}_i\bar{r}_i + X_iY_ir_i$$

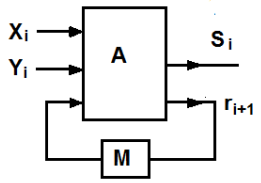
$$r_{r+1} = X_iY_i + X_ir_i + X_ir_i$$

si avrà il seguente schema:



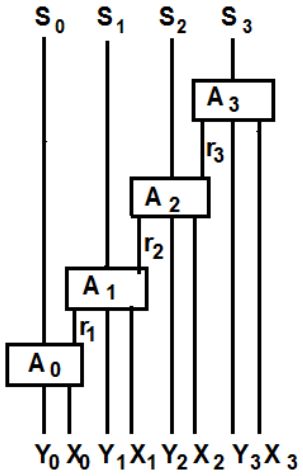
L'addizionatore elementare che realizza le funzioni S_i e r_{i+1} può essere utilizzato sia in un addizionatore in serie sia in un addizionatore in parallelo.

10.1.3 Addizionatore in serie-Addizionatore in parallelo



Nell'addizionatore in serie le cifre degli operandi X e Y (X_i, Y_i) si presentano negli istanti successivi I.

Una memoria M (per esempio una linea di ritardo con ritardo di un bit) conserva il riporto r_{i+1}, elaborato all'istante I, fino all'istante $i+1$ (vedi figura).



Nell'addizionatore in parallelo, le cifre X_i e Y_i $(i = 0, ..., n)$ degli operandi X e Y si presentano in entrata in parallelo (simultaneamente).

Con $n+1$ cifre, il circuito possiede $2(n+1)$ ingressi (X_i, Y_i) e $n+1$ uscite.

Nella versione più classica (forma iterativa) esso possiede $n+1$ blocchi o **stadi**: uno stadio di rango 0 che è un semiaddizionatore e n stadi costituiti da addizionatori elementari del tipo descritto in (2.1.2) (rango i> 0).

Il riporto elaborato al rango I (r_{i+1}) costituisce uno degli ingressi dello stadio $i+1$.

Nella figura posta accanto $n = 3$.

Si noterà che la lunghezza massima che i segnali devono attraversare è di $n+1$ stadi, ossia di $2(n+1)$ livelli se i blocchi sono appunto a 2 livelli.

Per certe condizioni ($X = 2^{n+1} - 1$, $Y = 1$ per es.) la lunghezza di propagazione del riporto è effettivamente uguale a questo valore.

Un addizionatore in parallelo completa l'addizione in un tempo che è determinato dal tempo di arrivo di un digit più il tempo di propagazione dell'eventuale riporto, mentre un addizionatore seriale richiede n intervalli di tempo (digit times). Quindi l'addizionatore parallelo ha il vantaggio di essere veloce, mentre quello seriale ha il vantaggio di avere meno circuiti.

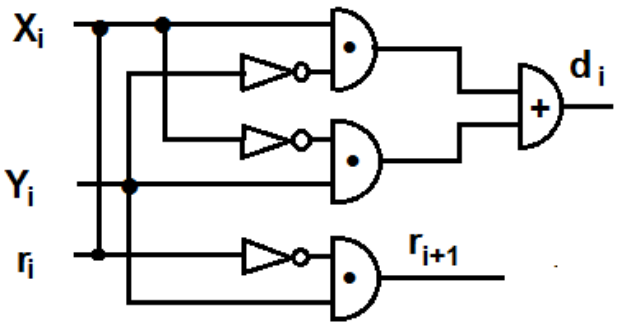
X_i	Y_i	d_i	r_{i+1}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

10.1.4 Sottrazione binaria

Mediante la tabella della verità a fianco si può definire il funzionamento di un blocco chiamato **semi-sottrattore** (half-subtractor) binario. Si fa l'ipotesi che il minuendo Xe il sottraendo Y siano positivi e tali che $X \geq Y$.

Le equazioni booleane che definiscono la differenza e la ritenuta in una sottrazione binaria sono dunque:

$$d_i = \bar{X}_i Y_i + X_i \bar{Y}_I = X_i \oplus Y_i$$
$$r_{i+1} = \bar{X}_i Y_i$$



Solo la seconda equazione differisce da quella ottenuta per l'addizione: essa si può dedurre da quella del par. 2.1.1 sostituendo X_i con $\bar{X}_i$.

Un semisottrattore genera la differenza d e la ritenuta r_{i+1} mediante il minuendo X_i e il sottraendo Y_i.

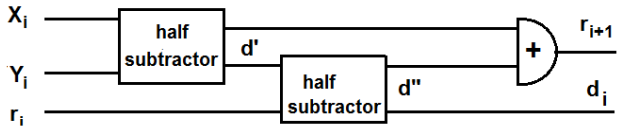
X_i	Y_i	r_i	d_i	r_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Volendo invece ottenere la differenza d_i e la ritenuta r_{i+1} considerando il sottraendo X_i, il minuendo Y_i e la ritenuta r_i del passo precedente, si ottiene la tabella della verità annessa al fianco.

Le equazioni booleane corrispondenti sono:

$$d_i = \bar{X}_i\bar{Y}_i r_i + \bar{X}_i y_i \bar{r}_i + X_i\bar{Y}_i\bar{r}_i + X_i Y_i r_i = X_i \oplus Y_i \oplus r_i$$
$$r_{i+1} = \bar{X}_i\bar{Y}_i r_i + \bar{X}_i Y_i\bar{r}_i + \bar{X}_i Y_i r_i + X_i Y_i r_i = \bar{X}_i Y_i + r_i(\bar{X}_i \oplus Y_i)$$

Uno dei possibili modi di realizzare un sottrattore elementare completo consiste nel considerare due semisottrattori.



Il semi-sottrattore e il sottrattore completo possono essere utilizzati per costruire sottrattori in serie e in parallelo, sempre con l'ipotesi che X e Y siano senza segno e con $X \geq Y$.

In seguito vedremo la realizzazione della sottrazione mediante l'uso dei complementi ad 1 e a 2. Infatti un circuito costruito per sommare numeri rappresentati con segno e modulo è molto più completo di quello che utilizza una rappresentazione complementata di numeri negativi.

10.2 Circuito di complementazione (ad 1)

Sia il numero $X = X_{n-1}X_{n-2}...X_0$, il suo complemento a 1 è il numero $C_1(X)$ che ha per rappresentazione $\bar{X}$:

$$C_1(X) = \bar{X} = \bar{X}_{n-1}\bar{X}_{n-2}...\bar{X}_k...\bar{X}_1\bar{X}_0$$

Si suppone X intero:

$$Es. \qquad X = 010110$$

$$C_1(X) = 101001$$

Si ha

$$X + C_1(X) = X + \bar{X} = 2^n - 1$$

Infatti, se si costruisce la somma, cifra dopo cifra, si ha:

$$\begin{cases} S_k = X_k \oplus \bar{X}_k \oplus r_k = r_k \oplus 1 = \bar{r}_k \\ r_{k+1} = X_k\bar{X}_k + r_k(X_k \oplus \bar{X}_k) = r_k \end{cases}$$

da cui, essendo $r_0 = 0$,

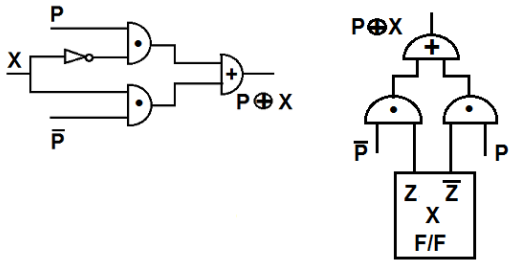
$$\begin{cases} S_k = 1 \\ r_h = 0 \end{cases}$$

per ogni k $(k = 0, ..., n-1)$

$$S = \overbrace{111...11}^{n\ cifre} = 2^n - 1$$

Un circuito complementatore (a 1) converte un numero binario avuto in input, in un output che ne è il complemento a 1.

Un complementatore può essere costruito in modo da ricevere un input seriale oppure uno in parallelo.



In figura sono mostrati due complementatori che forniscono in output il valore $P \oplus x$, essendo P il valore di un opportuno segnale di controllo: si ottiene il **complemento a 1** quando $P = 1$, infatti:

$$\bar{X} = X \oplus 1$$

se la variabile di controllo è $P = 0$ in output si ha la stessa variabile di ingresso X.

La figura a sinistra delle due mostra un complementatore seriale nel cui **input** X compare una sequenza binaria rappresentante un numero binario.

Questo circuito può essere usato come complementatore parallelo: è necessario allora avere un numero di circuiti pari al numero di bit del numero binario.

L'altra figura rappresenta un complementatore in parallelo: i due output del flip-flop X costituiscono i due input. Ovviamente ci devono essere tanti di questi circuiti quanti sono i bits del numero binario.

Questo circuito a sua volta può essere utilizzato come complementatore seriale: sarà allora richiesto un solo circuito e lo stato del flip-flop cambierà ad ogni digit time generando così una sequenza binaria rappresentante il numero binario.

10.3 Circuito di complementazione a 2

Sia il numero binario X, si chiama complemento vero o complemento a 2 la quantità $C_2(X) = 2^n - X$.

Si ha dunque:

$$C_2(X) = C_1(X) + 1$$

con

$$X = (X_{n-1}X_{n-2}...X_k...X_1X_0)$$

$$C_1(X) = (\bar{X}_{n-1}...\bar{X}_k...\bar{X}_0)$$

Aggiungendo 1 a $C_1(X)$ si ottiene:

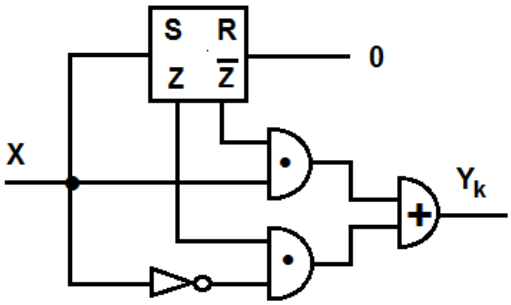
$$C_2(X) = (Y_n - 1Y_n - 2...X_k...Y_0) = Y$$

Ricordando come si esprimono in algebra booleana le somme e i riporti di una addizione dell'aritmetica booleana, si ha:

$$
\begin{aligned}
&Y_0 = \bar{X}_0 \oplus 1 \\
&Y_1 = \bar{X}_1 \oplus r_1 = \bar{X}_1 \oplus \bar{X}_0 \qquad\qquad r_2 = \bar{X}\bar{X}_0 \\
&X_2 = \bar{X}_2 \oplus r_2 = \bar{X}_2 \oplus \bar{X}_1\bar{X}_0 \qquad\qquad r_3 = \bar{X}_2\bar{X}_1\bar{X}_0 \\
&Y_k = \bar{X}_k \oplus \prod_{i=0}^{i=k-1} \bar{X}_i = X_k \oplus 1 \oplus \prod_{i=0}^{i=k-1} \bar{X}_i = \\
&\qquad X_k \oplus \overline{(\prod_{i=0}^{i=k-1} \bar{X}_i)} = X_k \oplus \sum_{i=0}^{i=k-1} X_i
\end{aligned}
$$

Ponendo:

$$Z_k = \sum_{i=0}^{i=k-1} X_i$$

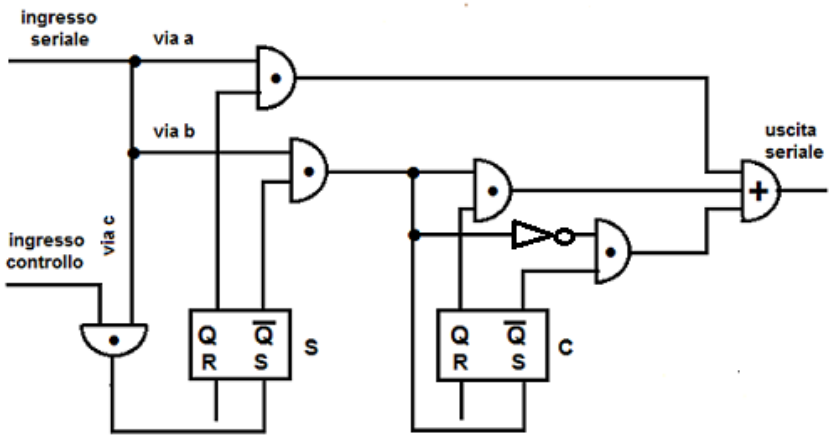


si ottengono le seguenti equazioni di un circuito complementare a 2 seriale:

$$
\begin{cases}
Y_k = X_k \oplus Z_k \\
Z_{k+1} = Z_k + X_k
\end{cases}
$$

Tenendo presente l'equazione booleana di un flip-flop SR, si può pensare di realizzare l'equazione $Z_{k+1} = Z_k + X_k$ mediante un SR con l'ingresso R sempre mantenuto a 0 e con lo stato interno iniziale $Z_0 = 0$.

Nella figura è descritto un complementatore a 2 di tipo seriale per numeri binari senza segno.



Nella seguente figura è descritto invece un complementatore seriale a 2 per numeri binari con segno: la sua caratteristica è determinata dal fatto che se il numero è negativo (bit del segno uguale ad 1), esso esce in output complementato a 2, se l'input invece è positivo, il numero uscirà nella forma vera.

Si fa l'assunzione che il bit del segno arrivi come primo bit nella sequenza di input: gli seguiranno i bit delle cifre significative in ordine di peso crescente.

Quando il bit del segno, arriva in input, esso passa attraverso la **via a** e compare invariato in output.

Nel frattempo un segnale di controllo del segno selezione la **via c**.

Se il bit del segno è 0 (numero positivo), il flip-flop S rimane nello stato 0 e i bit successivi del numero passano attraverso la **via a** ed appaiono in output.

Se il bit del segno è 1 (numero negativo), il flip-flop S viene messo nello **stato 1**. La **via a** viene interdetta e la **via b** viene permessa.

I bit successivi del numero passeranno allora per la **via b** e verranno complementati nella maniera descritta per numeri senza segno.

Osservando la figura del diagramma logico del complementate a 2 di tipo seriale si verifica la regola pratica per ottenere il complemento a 2 (detto anche complemento vero) di un numero binario. Essa consiste nel lasciare inalterati tutti gli zeri prima del primo 1 incontrato nel risalire la parola X partendo dai pesi più deboli e procedendo verso quelli più forti, nel lasciare inalterato questo 1, e nel complementare tutte le cifre a sinistra di questo 1.

$$\begin{aligned} \text{Esempio} \quad X &= 0110100 \\ C_2(X) &= 1001100 \end{aligned}$$

Se consideriamo il circuito dell'accennata figura notiamo che il flip-flop interviene in due porte di AND: in una i bits di X entrano in forma negata e nell'altra no.

Ora, consideriamo un input seriale a partire dai pesi più deboli, i bit 0 non cambiano lo stato iniziale del flip-flop la cui uscita Z=0 interdice la porta AND relativa negata dei bits di X, e questo fino all'arrivo del primo bit 1, dopo il quale lo stato del flip-flop cambia; si ha la situazione inversa della precedente, in cui l'uscita $\bar{Z} = 0$ interdisce la porta AND relativa ai bits di X.

Si ha inoltre:

$$X + C_2(X) = 2^n = 0 \ mod \ 2^n$$

Si vede quindi che, facendo le operazioni modulo 2^n, si ottengono gli stessi risultati sia operando con $X - Y$ che con $X + C_2(Y)$.

Quanto affermato sopra, è già stato dimostrato con l'ausilio del calcolo dei residui nel Cap.2 parte I.

Dimostriamo ora la veridicità mediante l'algebra di Boole. Poniamo:

$$C_2(Y) = Y' = (Y'_{n-1}Y'_{n-2}...Y'_0)$$

Le operazioni $X - Y$ e $X + C_2(Y) = X + Y'$ sono definite in forma ricorrente dalle equazioni:

$$(X - Y) \quad \begin{cases} S_k = X_k \oplus Y_k \oplus r_k \\ r_{k+1} = \bar{X}_k Y_k \oplus Y_k r_k \oplus \bar{X}_k r_k \end{cases}$$

$$(X + Y') \quad \begin{cases} S'_k = X_k \oplus Y'_k \oplus r'_k \\ r_{k+1} = X_k Y'_k \oplus Y'_k r'_k \oplus X_k r'_k \end{cases}$$

Supponiamo che al rango k si abbia $S'_k = S_k$. Si ha allora:

$$X_k \oplus Y_k \oplus r_k = X_k \oplus [Y_k \oplus (\sum_{i=0}^{i=k-1} Y_i)] \oplus r'_k$$

cioè

$$r'_k = r_k \oplus (\sum_{i=0}^{i=k-1} Y_i)$$

Si deduce immediatamente:

$$r'_{k+1} = X_k[Y_k \oplus (\sum_{i=0}^{k-1} Y_i)] \oplus [Y_k \oplus (\sum_{i=0}^{k-1} Y_i)] \cdot [r_k \oplus (\sum_{i=0}^{k-1} Y_i)] \oplus X_k[r_k \oplus (\sum_{i=0}^{k-1} Y_i)] =$$

$$X_k Y_k \oplus Y_k r_k \oplus X_k r_k \oplus (\sum_{i=0}^{k-1} Y_i)(Y_k \oplus r_k \oplus 1)$$

da cui

$$r'_{k+1} = r_{k+1} \oplus Y_k \oplus (\sum_{i=0}^{k-1} Y_i)(Y_k \oplus r_k \oplus 1 =$$

$$r_{k+1}[Y_k \oplus Y_k(\sum_{i=0}^{k-1} Y_i) \oplus (\sum_{i=0}^{k-1} Y_i] \oplus r_k(\prod_{i=0}^{k-1} \bar{Y}_i) =$$

$$r_{k+1} \oplus (\sum_{i=0}^{i=k} Y_i) \oplus r_k(\prod_{i=0}^{k-1} \bar{Y}_i)$$

Consideriamo l'ultimo termine. Si ha:

$$r_k(\prod_{i=0}^{k-1} \bar{Y}_i) = (\bar{x}_{k-1}Y_{k-1} \oplus Y_{k-1}r_{k-1} \oplus \bar{x}_{k-1}r_{k-1})(\prod_{i=0}^{k-1} \bar{Y}_i) =$$

$$\bar{X}_{k-1}\bar{Y}_{k-1}r_{k-1}(\prod_{i=0}^{k-2}\bar{Y}_i)$$

e quindi iterando il procedimento, si ha:

$$r_k(\prod_{i=0}^{k-1}\bar{Y}_i) = (\prod_{i=0}^{k-1}\bar{X}_i\bar{Y}_i)r_0 = 0 \quad in\ quanto\ r_0 = 0$$

Da cui:

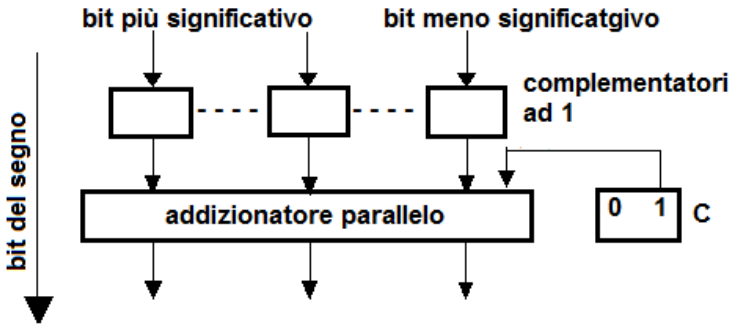
$$r'_{k+1} = r_{k+1} \oplus (\sum_{i=0}^{i=k} Yi)$$

di conseguenza

$$S'_{k+1} = S_{k+1}$$

E quindi se $S'_k = S_k$ si ha $S'_k + 1 = S_k + 1$

Poiché $S'_0 = S_0 = (X_0 \oplus Y_0)$, la relazione di uguaglianza sussiste per ogni rango K: $S'_k = S_k$.



Per quanto riguarda un complementatore a 2 parallelo, esso consiste in n complementatori ad 1, in un addizionatore parallelo ad n bit e in un flip-flop C. Con n si è indicato il numero dei bit che compongono il numero binario. Il bit del segno passa inalterato nel circuito. Per ottenere il complemento a 2 del numero, se ne trova prima il complemento ad 1 e quindi si somma 1 al bit meno significativo $[C_2(X) = C_1(X) + 1]$

L'addizione di 1 al bit meno significativo la si ottiene ponendo inizialmente il **flip-flop C** dello stato 11; in tal modo questa operazione consiste nell'addizionare due addendi di cui uno nullo, ma con un riporto iniziale di 1, e quindi sostanzialmente ogni adder dell'addizionatore parallelo lavora com un **half-adder**.

10.4 Controlli di parità

Si chiama **Bit di Parità**, una cifra binaria che si aggiunge talvolta ad una parola binaria in modo da scoprire eventuali errori di trasmissione.

Data una parola X ad n posizioni:

$$X = X_n X_{n-1} .... X_1$$

Si usa trasmettere la parola ad $n+1$ posizioni :

$$X' = X_n X_{n-1} .... X_1 P$$

dove P è tale che il numero totale degli 1 della parola X' sia pari.

Esempio:

$$\begin{cases} X = 11010110 \\ X' = 110101101 \end{cases}$$

$$\begin{cases} X = 10010110 \\ X' = 100101100 \end{cases}$$

Si può fare un'altra convenzione e definire un bit di disparità, tale che il numero totale degli 1 sia dispari.

Con questa nuova convenzione, non vi può essere un numero X' formato da zeri.

Questa combinazione risulta essere proibita, e la sua presenza può servire a mettere in evidenza certi difetti di funzionamento (come per esempio un difetto di alimentazione).

Si ha la corrispondenza:

$$P = 1, \ \sum_{i=1}^{n} X_i \text{ è dispari}$$

$$P = 0, \ \sum_{i=1}^{n} X_i \text{ è pari}$$

e quindi:

$$P = |\sum_{i=n}^{n} X_i|_2$$

dove P e X_i sono variabili aritmetiche. Con la convenzione già usata, considerando P e X_i variabili booleane, si ha:

$$P = X_i \oplus X_2 \oplus .... \oplus X_i \oplus .... \oplus X_n = \oplus \sum_{i=1}^{n} X_i$$

Il bit di parità indica gli errori di trasmissione su di un numero dispari di posizioni.

Come circuito di controllo, si può pensare un flip-flop di tipo T dove lo stato interno Q_0 iniziale è posto a zero.

$$P = Q_0 \oplus \sum_{i=1}^{n} x_i$$

10.4.1 Addizione di 2 numeri con bit di parità

Consideriamo 2 numeri binari X e Y ad n cifre:

$$\begin{cases} X = x_{n-1} x_{n-2} ... x_0 \quad P_0 \\ Y = y_{n-1} y_{n-2} ... y_0 \quad P_y \end{cases}$$

alle n cifre siano stati aggiunti i bit di parità P_x e P_y:

$$\begin{cases} P_x = \oplus \sum_{k=0}^{k=n-1} x_k \\ P_y = \oplus \sum_{k=0}^{k=n-1} y_k \end{cases}$$

chiamiamo P_s la cifra di parità $x + y$.

Calcoliamo l'espressione di P_s mediante le equazioni che danno la somma s_k e il riporto $r_k + 1$ di rango K in funzione di x_k, y_k, r_k.

Poniamo:

$$S = s_{n-1}s_{n-2}....s_0$$

(Supponiamo quindi che S sia anch'essa di n cifre).

Si ha allora:

$$\begin{aligned} P_s &= s_{n-1} \oplus s_{n-2} \oplus .... \oplus s_0 = \oplus \sum_{k=0}^{n-1} s_k \\ &= \oplus \sum_{k=0}^{n-1} (x_k \oplus y_k \oplus r_k) \\ &= (\oplus \sum_{k=0}^{n-1} x_k) \oplus (\oplus \sum_{k=0}^{n-1} y_k) \oplus (\oplus \sum_{k=0}^{n-1} r_k) \\ &= P_x \oplus P_y \oplus (\oplus \sum_{k=0}^{n-1} r_k) \qquad (1.7.1) \end{aligned}$$

La cifra di parità P_s di $S = X + Y$ è quindi la somma modulo 2 delle cifre dparità P_x e $P_y$di X e Y e dei riporti r_k $(k = 0, ..., n - 1)$.

Nota. Quando la cifra P_s calcolatamediane la somma effettivamente ottenuta e la cifra P_s calcolata mediante la (1.7.1) sono differenti, vi è errore. Poiché tutte le operazioni aritmetiche in un calcolatore sono effettuate, in ultima analisi, mediante addizioni, il procedimento sopra esposto può essere, in teoria, esteso come verifica di una qualunque operazione aritmetica.

10.5 Circuito per il confronto di 2 numeri binari

Il problema che ci si propone di risolvere è quello di effettuare una operazione di confronto tra due numeri binari ad n bit in modo da determinare quale dei due ha valore maggiore in modulo.

In un primo tempo costruiremo un comparatore seriale nel cui input arrivano i bits da confrontare secondo la sequenza determinata dal crescere del rango. Sia dunque la parola binaria:

$$X = x_{n-1}x_{n-2}....x_k.....x_1x_0$$

chiameremo con $X(k)$ la parola tronca formata dalle $k + 1$ cifre di peso più debole (ranghi da 0 a k).

$$X(k) = x_kx_{k-1}....x_0$$

Consideramo quindi due numeri X e Y e la variabile binaria definita dalla corrispondenza

S_k	x(k)-y(k)
0	≥ 0
1	< 0

S_{k-1}	x_k	v_k	S_k
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Da queste condizioni segue che:

$$\text{Se } x_k = y_k$$
$$\text{Allora } S_k = S_{k-1}$$

D'altra parte, se $x_k \neq y_k$, la diseguaglianza tra x_k e y_k da una parte, $x(k)$ $y(k)$ sono definite nello stesso verso.

Quanto detto può, può essere riassunto dalla tabella seguente.

Si ha dunque:

$$\begin{aligned} S_k = \bar{S}_{k-1}x_k\bar{y}_k + S_{k-1}\bar{x}_k\bar{y}_k + S_{k-1}x_k\bar{y}_k + S_{k-1}x_ky_k = \\ (x_k\bar{y}_k + x_ky_kS_{k-1}) + (\bar{x}_k\bar{y}_kS_{k-1} + x_k\bar{y}_k) = \\ x_k\bar{y}_k + x_kS_{k-1} + x_k\bar{y}_k + \bar{y}_kS_{k-1} = x_k\bar{y}_k + x_kS_{k-1} + \bar{y}_kS_{k-1} \end{aligned}$$

Si riconosce per S_k l'espressione della ritenuta ottenuta durante l'operazione $Y - X$.

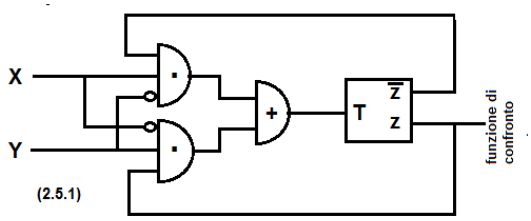
$$S_k = \bar{S}_{k-1}x_k\bar{y}_k + S_{k-1}\bar{x}_k\bar{y}_k + S_{k-1}x_k\bar{y}_k + S_{k-1}x_ky_k =$$

Si vede quindi che $S_n - 1 = 1$ quando l'operazione $Y - X$ produce una ritenuta al rango $n - 1$, situazione che si deve verificare quando $x > y$. Per confrontare i due numeri però non è necessaqrio determinare la differenza: infatti è sufficiente il valore $S_n - 1$ della ultima ritenuta di questa differenza. Per costruire il circuito, conviene però mettere la equazione.

$$S_k = \bar{S}_{k-1}x_k\bar{y}_k + S_{k-1}\bar{x}_k\bar{y}_k + S_{k-1}x_k\bar{y}_k + S_{k-1}x_ky_k$$

sotto la forma seguente:

$$
\begin{aligned}
S_k &= \bar{S}_{k-1}x_k\bar{y}_k \oplus S_{k-1}\bar{x}_k\bar{y}_k \oplus S_{k-1}x_k\bar{y}_k \oplus S_{k-1}x_ky_k = \\
&= \bar{S}_{k-1}x_k\bar{y}_k \oplus S_{k-1}\bar{x}_k(y_k \oplus 1) \oplus S_{k-1}x_k(y_k \oplus 1) \oplus S_{k-1}x_ky_k = \\
&= \bar{S}_{k-1}x_k\bar{y} : k \oplus S_{k-1}\bar{x}_k y : k \oplus S_{k-1} = \\
&= (\bar{S}_{k-1}x_k\bar{y}_k \oplus S_{k-1}\bar{x}_k y_k) \oplus S_{k-1} = \\
&= (\bar{S}_{k-1}x_k\bar{y}_k + S_{k-1}\bar{x}_k y_k) \oplus S_{k-1}
\end{aligned}
$$



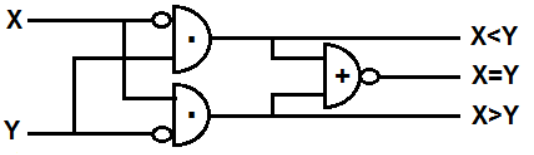
Per realizzare questo circuito, è sufficiente considerare un flip-flop di tipo T in cui $S_k - 1$ rappresenta lo stato interno ed il resto dell'espressione costituisce l'ingresso del flip-flop mentre l'uscita determina la funzione di confronto tra i due numeri X e Y.

L'operazione realizzata da l circuito ha termine ovviamente quando in input si sono presentati tutti gli n bits di X e Y.

X	Y	X=Y	X>Y	X<Y
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

Supponiamo invece che i bits di X e Y si presentino nell'ordine determinato dai pesi decrescenti: il circuito in questo caso sarà molto più semplice in quanto non ci sarà bisogno di utilizzare un circuito sequenziale.

La tabella di un comparatore seriale in cui l'input sia costituito da bits di peso decrescente sarà come quella affiancata.



Da tale tabella si deduce il circuito a lato.

Appena $x_k \neq y_k$ si arrestano i confronti.

10.5.1 Comparatore parallelo

Un comparatore parallelo sarà costituito da n circuiti comparatori in cui gli n bitsdi X e Y verranno confrontati separatamente e contemporaneamente.

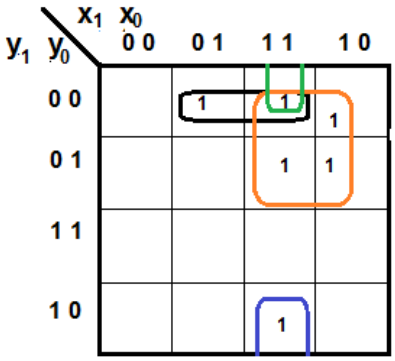
La prima coppia di bits x_i e y_i (i=n-1,n-2,...,0) che differisce, determina l'uscita finale del comparatore, a prescindere dal valore di $x_i - 1$, $y_i - 1$,...ecc.

Nel seguito viene costruito un circuito comparatore in parallelo di 2 numeri di 2 bit: un circuito relativo a parole a n bits, avrà l'uscita uguale a quello di figura 2.5.2 e 2 circuiti $NAND$ in più per ogni coppia di bit addizionale.

Siano quindi da confrontare i due numeri; $X = x_1x_0$ e $Y = y_1y_0$.

La tabella della verità relativa al problema del confronto in parallelo sarà:

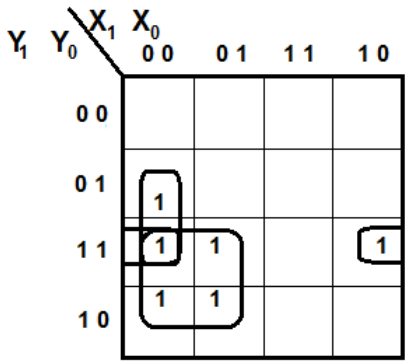
$1) X > y \quad S_1 = \bar{x}_1x_0\bar{y}_1\bar{y}_0 + x_1\bar{x}_0\bar{y}_1\bar{y}_o + x_1\bar{x}_0\bar{y}_1y_0 + x_1x_0\bar{y}_1\bar{y}_0 + x_1x_0y_1\bar{y}_1y_0 + x_1x_0y_1\bar{y}_0$



Semplificando la funzione mediante un diagramma di Karnaugh si ha:

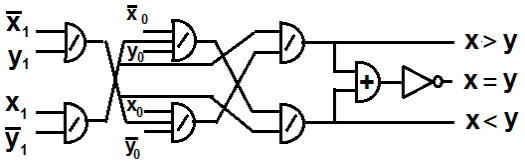
$$\begin{aligned} S_1 &= x_1\bar{y}_1 + x_0\bar{y}_1\bar{y}_0 + \bar{y}_0x_1x_0 = \\ &= \overline{\overline{x_1\bar{y}_1 + x_0\bar{y}_0(x_1 + \bar{y}_1)}} = \\ &= \overline{\overline{x_1\bar{y}_1} \cdot \overline{x_0\bar{y}_0(x_1 + \bar{y}_1)}} = \\ &= \overline{\overline{x_1\bar{y}_1} \cdot \overline{x_0\bar{y}_0(\bar{x}_1 \cdot y_1)}} \end{aligned}$$

$2) -x + y \quad S_2 = \bar{x}_1\bar{x}_0\bar{y}_1y_0 + \bar{x}_1\bar{x}_0y_1\bar{y}_0 + \bar{x}_1\bar{x}_0y_1y_0 + \bar{x}_1x_0y_1\bar{u}_0 + \bar{x}_1x_0y_1y_0 + x_1\bar{x}_0y_1y_0$



Semplificando la funzione mediante un diagramma di Karnaugh si ha:

$$\begin{aligned} S_2 &= \bar{x}_1 y_1 + \bar{x}_0 y_1 y_0 + \bar{x}_1 \bar{x}_0 y_0 = \\ &= \overline{\overline{\bar{x}_1 y_1 + y_0 \bar{x}_0 (y_1 + \bar{x}_1)}} = \\ &= \overline{\overline{\bar{x}_1 y_1} \cdot \overline{y_0 \bar{x}_0 (y_1 + \bar{x}_1)}} = \\ &= \overline{\overline{\bar{x}_1 y_1} \cdot \overline{y_0 \bar{x}_0 (\bar{y}_1 \cdot x_1)}} \end{aligned}$$



$$3) - \quad X = Y$$

Dalla tabella della verità si deduce che:

$$S_3 = \overline{S_1 + S_2}$$

Utilizzando alloa delle porte di $NAND(/)$ si costruirà il circuito a latere

10.6 Convertitore codice binaio puro-binario riflesso e viceversa

Sia il numero binario X e siano le sue due rappresentazioni binarie:

- in binario puro X sia $B = (B_n B_{n-1} .... B_0)$
- in binario riflesso X sia $R = R_n R ... R_0)$.

Le formule di conversione (Cap.3 parte I) sono:

$$(R \longrightarrow B): \qquad \oplus B_p = \sum_{j=p}^{j=n} R_j$$

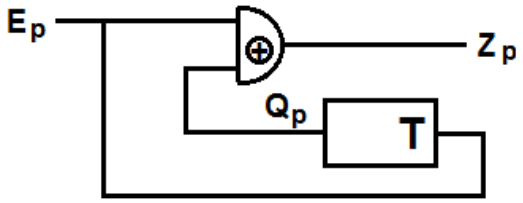
$$(B \longrightarrow R): \qquad R_p = B_p \oplus B_{p+1} \; ; \; R_n = B_n$$

Le conversioni si fanno più facilmente dalle cifre di peso più forte.
In queste condizioni, poniamo, per la conversione binario riflesso-binario puro:

$$\begin{cases} Z_p = B_{n-p} \\ Q_p = Bn - p - 1 \qquad (p \geq 1), Q_B = 0 \\ E_p = R_{n-p} \end{cases}$$

Si ha:

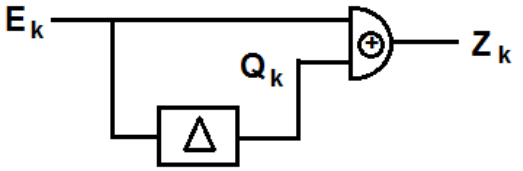
$$\begin{cases} Q_p = Q_p \oplus E_p \\ Z_p = Q_p \oplus E_p \end{cases}$$



Mediante le equazioni trovate possiamo dedure lo schema del circuito che realizza la conversione da un codice riflesso ad uno binario puro: per avere la uscita Z_p è sufficiente una porta di XOR mentre per l'uscita $Q_p + 1$ è necessario utilizzare un flip-flop T.

Per la conversione binario puro-binario riflesso, si pone:

$$\begin{cases} Q_k = B_{n-k+1} \\ E_k = B_{n-k} \\ Z_k = R_{n-k} \end{cases}$$



si ha quindi

$$\begin{cases} Q_{k+1} = E_k \\ Z_k = Q_k \oplus E_k \end{cases} \qquad con\ Q_0 = 0$$

Da cui il circuito nella figura accanto:

10.7 Contatori

Si chiama contatore un circuito capace di prendere nota del numero di impulsi che arrivano in **input** e conservare il risultato dell'operazione di conteggio.

Se un contatore somma ogni qual volta arriva un impulso in input, esso pende il nome di **forward counter** o **up counter**. Se un contatore assume inizialmente una certa configurazione rappresentante un numero ed ad ogni impulso in input sottrae da questo numero, esso viene chiamato **reverse counter** o **down counter**.

Se un contatore è provvisto di due terminali di input in modo da addizionare se gli impulsi arrivano ad uno dei terminali di input e da sottrarre per gli impulsi che arrivano all'altro input, esso viene chiamato **reversible counter** o **up-down counter**.

I contatori vengono classificati anche in base alla maniera di memorizzare il numero del conteggio.

Se il numero memorizzato è un numero binario, in codice binario puro il contatore è un contatore binario.

Se il numero memorizzato è un numero decimale è un contatore decimale.

Se il digit decimale viene rappresentato mediante un codice binario il contatore è un **binay-codet decimal digit counter**.

Vi sono inoltre contatori capaci di esprimere il conteggio in altri codici binari (p.es. codice Gray). Pe ottenere le equazioni booleane di un contatore binario puro si può pocedee in due modi.

10.7.1 Primo modo per la determinazione booleana di un contatore binario

Incremento in codice binario puro

$$\begin{array}{l} X = x_n\, x_{n-1} \ldots x_0 + \\ +1 = 0\ \ 0 \ldots\ldots 1 \\ \hline X' = x'_n\, x'_{n-1} \ldots x'_0 \end{array}$$

Il passaggio da un numero X al numero $X+1$ può essere rappresentato mediante certe relazioni booleane che legano la rappresentazione dei numeri X e $X+1$. Consideriamo infatti l'addizione:

Si ha:

$$x'_0 = x_0 \oplus 1 = \bar{x}_0 \qquad r_1 = x_0 \cdot 1$$

$$x'_1 = x_1 \oplus r_1 = x_1 \oplus x_0 \qquad r_2 = x_1 x_0$$

$$x'_2 = x_2 \oplus r_2 = x_2 \oplus x_1 x_0 \quad r_3 = x_2 x_1 x_0$$

$$- - - - - - - - - - - - \qquad - - - - -$$

$$x'_k = x_k \oplus r_k = x_k \oplus \coprod_{i=0}^{i=k-1} x_i \ ; \ r_{k+1} = \coprod_{i=0}^{i=k} x_i$$

$$\begin{cases} x'_k = x_k \oplus \coprod_{i=0}^{i=k-1} x_i \\ r_{k+1} = \coprod_{i=0}^{i=k} x_i \end{cases}$$

Le relazioni che rappresentano il numero $x+1$ in funzione di X sono:

$$\begin{cases} x'_k = x_k \oplus \coprod_{i=0}^{i=k-1} x_i \\ r_{k+1} = \coprod_{i=0}^{i=k} x_i \end{cases} \qquad (2.7.2)$$

Decremento in codice binario puro

Le relazioni che danno la rappresentazione del numero $X-1$ in funzione di quella di X possono essere ottenute con il seguente procedimento:

$$\begin{array}{l} X = x_n\ \ x_{n-1} \cdots x_0 \\ -1 = 0\ \ \ 0 \ldots\ldots 1 \\ \hline X'' = x''_n\ \ x''_{n-1} \ldots x''_0 \end{array}$$

$$
\begin{array}{ll}
x_0'' = x_0 \oplus 1 = \bar{x}_0 & r_1 = \bar{x}_0 \\
x_1'' = x_1 \oplus r_1 = x_1 \oplus \bar{x}_0 & r_2 = \bar{x}_1 r_1 = \bar{x}_1 \bar{x}_0 \\
x_2'' = x_2 \oplus r_2 = x_2 \oplus \bar{x}_1 \bar{x}_0 & r_3 = \bar{x}_2 r_2 = \bar{x}_2 \bar{x}_1 \\
\hline
x_k'' = x_k \oplus r_k = x_k \oplus \coprod_{i=0}^{i=k-1} \bar{x}_i; & r_{k+1} = \coprod_{i=0}^{i=k} \bar{x}_i
\end{array}
$$

$$
\begin{cases} x_k{}'' = x_k \oplus \coprod_{i=0}^{i=k-1} \bar{x}_i \\ r_{k+1} = \coprod_{i=0}^{i=k} \bar{x}_i \end{cases} \qquad (2.7.3)
$$

Le relazioni (2.7.2) rappresentano il meccanismo di un **u-counter** e le (2.7.3) quelle di un **down-counter**, entrambi un codice binario puro. Si noterà che si passa dall'incremento al decremento complementando le variabili che compaiono nella ritenuta.

Complementando ambo i membri della relazione (2.7.3) si ottiene:

$$\bar{x}_k'' = \bar{x}_k \oplus \coprod_{i=0}^{k-1} \bar{x}_i$$

che insieme alla:

$$r_{k+1} = \coprod_{i=0}^{k} \bar{x}_i$$

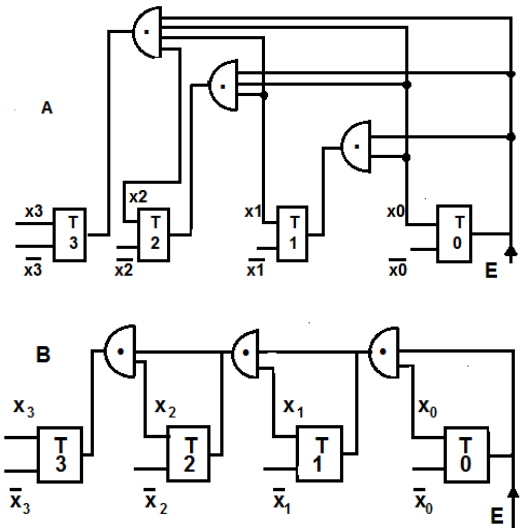
dà la relazione di incremento per i numeri corrispondenti alle parole binarie complementate. Infatti chiamiamo $|A|$ il valore rappresentato dal nome A, si ha:

$$
\begin{gathered}
|x'| = |x| + 1 \\
|x^{''}| = |x| - 1 \\
|\bar{x}'| = 2^n - 1 - |x'| = 2^n - 1 - |x| - 1 = |\bar{x}| - 1 \\
|\bar{x}^{''}| = 2^n - 1 - |x^{''}| = 2^n - 1 - |x| + 1 = |\bar{x}| + 1
\end{gathered}
$$

Se si chiama con E la variabile d'entrata del circuito, con l'indice n si indica il tempo dell'orologio ($n \geq 0$), si ottengono le seguenti equazioni:

$$\text{up-counter:} \qquad x_k^{n+1} = x_k^n \oplus E^n (\coprod_{i=0}^{k-1} x_i)_n$$

$$\text{down-counter:} \qquad x_k^{n+1} = x_k^n \oplus E^n (\coprod_{i=0}^{k-1} \bar{x}_i)_n$$



Le figure A e B rappresentano degli **up-conter** a 16 stadi interni quattro stadi binari). L'orologio non viene indicato).

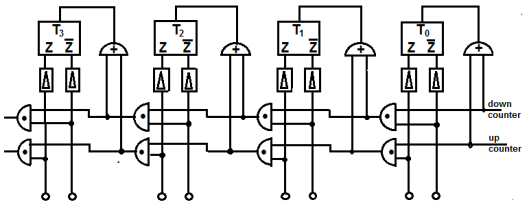
La generalizzazione al caso di m stadi (2^m stadi interni) è immediata.

Si noterà che per m elevato non è possibile connettere i flip-flop mediante una logica ad 1 livello come in figura A in quanto per il rango k è necessari una porta a k entrate.

Si possono allora realizzare i prodotti πx_i mediante funzioni di funzioni come in figura B.

Nella configurazione data, sono sufficienti delle porte AND a due entrate: risulta essere più lungo però il tempo di propagazione del riporto.

Per il rango k ci vuole un tempo k volte maggiore di quello necessario per il contatore A.



Le cifre x_k di rango k ad un certo istante n sono ovviamente rappresentate dagli stati interni dei flip-flop T di rango corrispondente mentre le uscite complimentate rappresentano le $\bar{x}_k$."'

Le x_k fanno parte della rappresentazione di $x + 1$ mentre le $\bar{x}_k$ sono la rappresentazione binaria di $\bar{x} - 1$.

La costruzione di un **down-counter** risulta ovvia e la si può vedere facilmente nel grafico sovrastante dove e rappresentato un **up-down counter**

Nello schema sono state aggiunte delle linee di ritardo: esse stanno ad indicare che l'input delle porte di AND che proviene dai flip-flop è lo stato dei flip-flop prima che essi vengano commutati.

Se i flip-flop posseggono un loro ritardo sufficiente, questi ritardi aggiuntivi non sono necessari.

Come si è appunto supposto nelle figure A e B).

Il disegno logico di un contatore binario può essere ricavato utilizzando una tavola della verità. La tavola mostra non solo le relazioni tra i successivi stati dei flip-flop ma anche gli stati di input dei flip-flop che causano una commutazione di questi Flp-Flop. Consideriamo un up-counter binario a tre stadi e proponiamoci di costruirlo con flip-flop di tipo T.

La tavola della verità di questo contatore è mostrata nella figura D.

In questa tavola i tre flip-flop e le loro uscite sono rappresentati con le variabili A_3, A_2, A_1 rappresentano i loro rispettivi **input**; p rappresenta gli impulsi in input da conteggiare; t_0, t_1 ecc. rappresentano la sequenza temporale degli impulsi.

Le tre colonne relative agli A mostrano la sequenza desiderata degli stati dei tre flip-flop. Le ultime tre colonne mostrano i valori veri richiesti per gli **input** a_{3t}, a_{2t}, a_{1t}, capaci di cambiare lo stato del contatore.

In queste tte colonne, lo 0 significa che non è richiesto alcun impulso nell'input del flipo-flop, mentreun 1 significa che è necessario un impulso nell'input del flip-flop.

Questi 0 e questi 1 sono ottenuti dalla commutazione dello stato dei flip-flop indicata nelle tre colonne relative agli A.

tempo	p	A_3	A_2	A_1	a_{3t}	a_{2t}	a_{1t}
T_0	1	0	0	0	0	0	1
T_1	1	0	0	1	0	1	1
T_2	1	0	1	0	0	0	1
T_3	1	0	1	1	1	1	1
T_4	1	1	0	0	0	0	1
T_5	1	1	0	1	0	1	1
T_6	1	1	1	0	0	0	1
T_7	1	1	1	1	1	1	1
T_8	1	0	0	0			
D							

Per esempio, quando lo stato del contatore viene cambiato da 000 a 001, gli stati di A_3 e A_2 non cambiano, quindi i valori di a_{3t} e a_{2t} sono entrambi 0 nella prima riga delle colonne relative a a_{3t} e a_{2t}.

Però lo stato di A_1 cambia da 0 a 1; quindi il valore di a_{1t} è 1 nella prima riga dell'ultima colonna. Gli altri valori delle ultime tre colonne vengono ottenuti nella stessa maniera.

Da notare che all'istante t_8 lo stato del contatore è 000 lo stesso che all'istante t_0.

Il contatore quindi ha un compotamednto ciclico.

Le equazioni di input per i tre flip-flop dedotte dalla tabella della verità D sono:

$$
\begin{aligned}
a_{1t} &= p \\
2_{2t} &= A_1 \cdot p \\
3_{3t} &= A_2 \cdot A_1 \cdot p
\end{aligned}
$$

che possono essere anche scritte come:

$$
\begin{aligned}
a_{1t} &= p \\
2_{2t} &= A_1 \cdot a_{1t} \\
3_{3t} &= A_2 \cdot a_{2t}
\end{aligned}
$$

Quando le equazioni di input vengono sostituite nell'equazione di stato di un flip-flop T ($Q_{n+1} = Q_n \oplus E_n$) si ottiene:

$$
\begin{aligned}
A_1(\pi) &= A_1 \oplus p \\
A_2(\pi) &= A_2 \oplus A_1 p \\
A_3(\pi) &= A_3 \oplus A_2 A_1 p
\end{aligned}
$$

Queste sono le equazioni di stato di un contatore e descrivono lo stato dei tre flip-flop.

	tempo	p	A_3	A_2	A_1	A_{3r}	A_{3s}	A_{2r}	A_{2s}	A_{1r}	A_{1s}
	t_0	1	0	0	0	0	0	0	0	0	1
	t_1	1	0	0	1	0	0	0	1	1	0
	t_2	1	0	1	0	0	0	0	0	0	1
	t_3	1	0	1	1	0	1	1	0	1	0
E	t_4	1	1	0	0	0	0	0	0	0	1
	t_5	1	1	0	1	0	0	0	1	1	0
	t_6	1	1	1	0	0	0	0	0	0	1
	t_7	1	1	1	1	1	0	1	0	1	0
	t_8	1	0	0	0						

Applicando queste equazioni a quelle precedenti di **input** si ottengono i circuiti di fig.A e B.

Al posto di un flip-flop T, si può pensare di utilizzare un SR per costruire un contatore binario. Nella tavola E compaiono le condizioni necessarie per costruire un **up-counter** binario mediante flip-flop SR.

In questa tabella a_{3r}, a_{2r}, a_{1r}, rappresentano gli input di riset (R) nei tre flip-flop ed a_{3s}, a_{2s}, a_{1s}, i tre set inputs, p rappresenta gli impulsi di input da conteggiare; t_0, t_1, ecc. rappresentano la sequenza temporale degli impulsi di input.

Gli 0 e gli 1 delle ultime sei colonne sono anche qui ottenuti mediante i cambiamenti di stato dei flip-flop indicati nelle tre prime colonne. Essi però differiscono da quelli della tabella D. Nella tabella E l'1 nella colonna dell' **input reset** (colonne sei, otto, dieci) indica il cambiamento di stato del flip-flop da 1 a 0, mentre lo 0 indica tutti gli altri casi (da 0 a 0, da 1 a 1, da 0 a 1).

L'1 nella colonna dell' **input reset** (colonne sette, nove, undici) indica il cambiamento dello stato del flip-flop da 0 a 1 mentre lo 0 indica tutti gli altri casi (da 0 a 0, da 1 a 1, da 1 a 0). Per esempio, quando lo stato del contatore cambia da 000 a 001, gli stati di A_3 e A_2 rimangono immutati, per cui i valori di a_{3r}, a_{3s}, a_{2r}, a_{2s} sono 0 nella prima riga delle colonne sei, sette, otto e nove.

Lo stato A_1 però deve cambiare da 0 a 1, quindi il valore di a_{1r} è 0 mentre il valore di a_{1s} è 1 nella prima riga delle due ultime colonne rispettivamente. I valori degli altri input **set** e **reset** sono ottenuti nella stessa maniera.

Dalla tabella E possiamo ottenere le equazioni di input dei tre flip-flop:

$$a_{1s} = \bar{A}_1 \cdot p$$

$$a_{2s} = \bar{A}_2 \cdot A_1 \cdot p = \bar{A}_2 \cdot a_{1r}$$

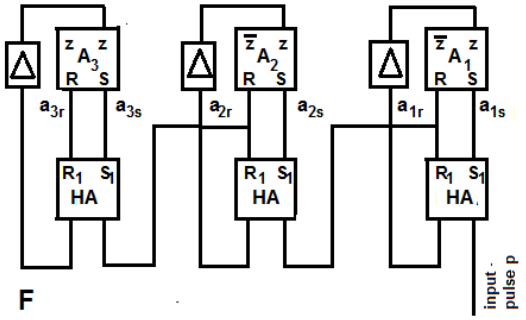
$$a_{3s} = \bar{A}_3 \cdot A_2 \cdot A_1 \cdot p = \bar{A}_3 \cdot a_{2r} \qquad 2.7.10$$

$$a_{1r} = A_1 \cdot p$$

$$a_{2r} = A_2 \cdot A_1 \cdot p = A_2 \cdot a_{1r}$$

$$a_{3r} = A_3 \cdot A_2 \cdot A_1 \cdot p = A_3 \cdot a_{2r}$$

Se si sostituiscono le equazioni di input nella equazione di stato di un flip-flop SR ($Q_{n+1} = Q_n\bar{R}_n + S_n$), si ottengono le equazioni **2.7.8**, cioè le equazioni di stato di un **up-counter** binario.



Le equazioni 2.7.10 si possono scrivere in un altro modo.

Alle prime tre equazioni di 2.7.10 si aggiungono rispettivamente i termini $A_1 \cdot \bar{p}, A_2 \cdot \bar{a}_{1r}, A_3 \cdot \bar{a}_{2r}$; queste somme logiche non cambiano il valore delle espressioni di partenza.

Le sei equazioni diventano allora:

$$a_{1s} = A_1 \oplus p \qquad a_{1r} = A_1 \cdot p$$

$$a_{2s} = A_2 \oplus a_{1r} \qquad a_{2r} = A_2 \cdot a_{1r}$$

$$a_{3s} = A_3 \oplus a_{2r} \qquad a_{3r} = A_3 \cdot a_{2r}$$

Si noti che queste sono le equazioni delle somme e del riporto in output di un **half-adder**.

Quindi a_{1s} e a_{1r} sono rispettivamente la somma e il riporto di un **half-adder** i cui input sono A_1 e p.

In modo equivalente, si ha che a_{2s} e a_{2r} come a_{3s} e a_{3r} sono le uscite di **half-addeers**.

Il contatore binario utilizzante degli **half-adder** e dei flip-flop SR, viene mostrat in figura F.

10.8 Contatori decimali

I contatori decimali sono circuiti che accettano degli impulsi in input, li contano e ogni volta che la somma totale risulti essere 10, ritornano allo stato iniziale emettendo sulla linea di uscita un impulso di riporto (dettom carry o overflow).

Tali contatori di capacità 10 vengono chiamati **decadi**.

Interconnettendo m elementi decimali di conteggio di questo tipo, in modo che l'uscita di un elemento $J-1$ serva come ingresso all'elemnto J, si può realizzare un contatore di capacità 10^m.

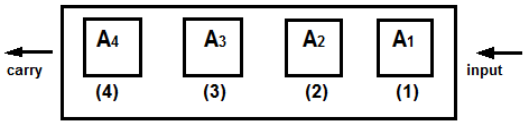
Per codificare i 10 stati interni si usano generalmente codifiche di tipo binario con almeno 4 digit binari.

	I	II
Q_0	0000	0011
Q_1	0001	0100
Q_2	0010	0101
Q_3	0011	0110
Q_4	0100	0111
Q_5	0101	1000
Q_6	0110	1001
Q_7	0111	1010
Q_8	1000	1011
Q_9	1001	1100

Si è visto (cap. III, parte 1) che quando la cifra decimale corrispondente deve essere convertita in analogico, può essere utile considerare un codice ponderato. Infatti nel caso di una conversione in tensione sarà sufficiente fare la somma ponderata delle correnti corrispondenti ai 4 digit binari.

Nella tabella posta accanto compaiono il codice 8421 (I) e quello ad **eccesso di 3** (II).

Ci proponiamo nel seguito di costruire un contatore relativo al codice 8421 e quello relativo al codice **ad eccesso di tre**.



La configurazione generale di un contatore decimale relativo al codice 8421 sarà:

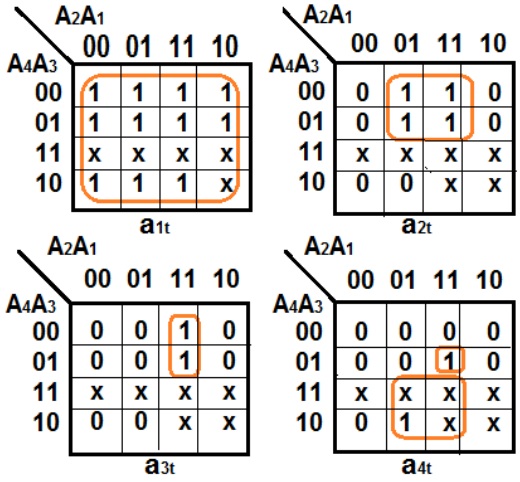
Il numero rappresentato dallo stato interno dei flip-flop A_4, A_3, A_2A_1, raggiungeà al massimo 1001, equivalente al digit decimale 9.

Appena arriva un nuovo impulso, il contatore assume lo stato 0000 (equivalente al digit decimale 0), e produce un riporto che servirà come input per incrementare la seconda decade.

p	stato	A_4	A_3	A_2	A_1	A_{4t}	A_{3t}	A_{2t}	A_{1t}
1	0	0	0	0	0	0	0	0	1
1	1	0	0	0	1	0	0	1	1
1	2	0	0	1	0	0	0	0	1
1	3	0	0	1	1	0	1	1	1
1	4	0	1	0	0	0	0	0	1
1	5	0	1	0	1	0	0	1	1
1	6	0	1	1	0	0	0	0	1
1	7	0	1	1	1	1	1	1	1
1	8	1	0	0	0	0	0	0	1
1	9	1	0	0	1	1	0	0	1

Queste considerazioni e la conoscenza delle proprietà dei flip-flop T che vogliamo utilizzare per realizzare il circuito, ci permettono di scrivere ls seguente tabella (2.8.2):

Possiamo scrivere le equazioni di input $a_{4t}, a_{3t}, a_{2t}, a_{1t}$ direttamente dalla tabella 2.8.2.



Possiamo semplificare le espressioni booleane mediante diagrammi di Karnaugh; in questa semplificazione si può far uso anche degli stati non esistenti 1010, 1011, 1100, 1101, 1110, 1111 (segnati con crocetta).

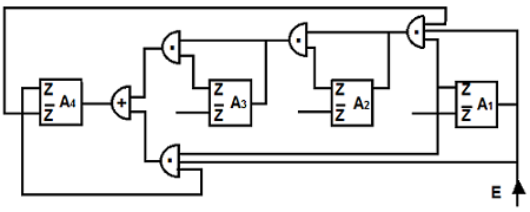
La variabile p è una variabile d'ingresso ma poiché vale sempre 1, è inutile prenderla in considerazione durante la semplificazione.

Dai digrammi si esplicano:

$$a_{1t} = p$$
$$a_{2t} = A_1\bar{A}_4 \cdot p$$
$$a_{3t} = A_1A_2\bar{A}_4 \cdot p$$
$$a_{4t} = \bar{A}_4A_3A_2A_1 \cdot p + A_4A_1 \cdot p$$

Sostituendo questi valori di input nella equazione di stato di un flip-flop si ottengono le equazioni di stato di una decade relativa al codice 8421:

$$A_1() = A_1 \oplus p$$
$$A_2() = A_2 \oplus A_1\bar{A}_4p$$
$$A_3() = A_3 \oplus A_1A_2\bar{A}_4p$$
$$A_4() = A_4 \oplus (\bar{A}_4A_3A_2A_1p + A_4A_1p)$$



Nella figura viene dato il diagramma logico di una realizzazione di questo circuito.

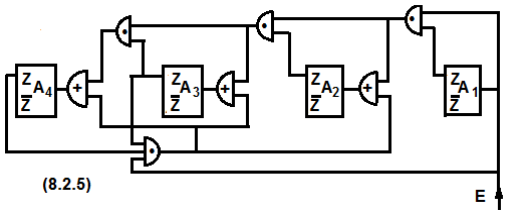
Supponiamo ora di voler costruire un contatore per un codice a accesso di tre.

La tabella della verità relativa viene data in figura 2.8.4

Semplificando con Karnaugh si ottengono le equazioni di input seguenti:

p	Stato	A_4	A_3	A_2	A_1	a_{4t}	a_{3t}	a_{2t}	a_{1t}
1	0	0	0	1	1	0	1	1	1
1	1	0	1	0	0	0	0	0	1
1	2	0	1	0	1	0	0	1	1
1	3	0	1	0	1	0	0	1	1
1	4	0	1	1	1	1	1	1	1
1	5	1	0	0	0	0	0	0	1
1	6	1	0	0	1	0	0	1	1
1	7	1	0	1	0	0	0	0	1
1	8	1	0	1	1	0	1	1	1
1	9	1	1	0	0	1	1	1	1

$$a_{1t} = p$$
$$a_{2t} = A_1 p + A_4 A_3 p$$
$$A_{3t} = a_1 a_2 P + a_4 a_3 p$$
$$a_{4t} = A_1 A_2 A_3 p + A_4 A_3 p$$



Per le semplificazioni si sono considerate anche le combinazioni che non compaiono nel codice ad eccesso di tre.

Dalle equazioni di input si deduce una possibile realizzazione del circuito (2.8.5): decadi per codici ad eccesso di tre.

10.9 Contatori in codice binario riflesso

Determiniamo per prima cosa le relazioni che legano due parole codice y_i e y_{i+1} rappresentanti in codice rilflesso due numeri x_i e x_{i+1} consecutivi, cioè tali che $x_{i+1} - x_i = 1$.

Si è visto (cap.3 parte I) che y_1 e y_{i+1} devono essere adiacenti.

Si possono avere due casi e si può scrivere così.

■ Primo caso: x_i pari

Si ha allora:

$$\begin{cases} x_i = x_n x_{n-1}..x_k..x_1 0 \\ e\ quindi \\ x_{i+1} = x_n x_{n-1}..x_k..x_1 1 \end{cases}$$

a questi due numeri corrispondono le seguenti rappresentazioni in codice Gray:

$$y_i = x_n(x_{n-1} \oplus x_n)(x_{n-2} \oplus x_{n-1})..(x_k \oplus x_{k+1})..(x_1 \oplus x_2)(0 \oplus x_1)$$
$$\bar{y}_{i+1} = x_n(x_{n-1} \oplus x_n)(x_{n-2} \oplus x_{n-1})..(x_k \oplus x_{k+1})..(x_1 \oplus x_2)(1 \oplus x_1)$$

Si ha quindi:

$$y_i \oplus y_{i+1} = 000..0..01 \qquad (2.9.1)$$

Quindi se y_i rappresenta in codice Gray un intero pari, si ottiene y_{i+1}, complemetando il digit più a destra.

■ Secondo caso: x_i dispari:

$$x_i$$

comincia a destra con un 1 e si può scrivere così:

$$x_i = x_n x_{n-1}..x_{k+1}..0\ 1\ 1..1\ 1$$
$$x_{i+1} = x_n x_{n-1}..x_{k+1}..1\ 0\ 0..0\ 0$$

Da cui:

$$y_i = x_n(x_{n-1} \oplus x_n)..(x_{k+1} \oplus x_{k+2})x_{k+1}1\ 0\ 0..0\ 0$$
$$y_{i+1} = x_n(x_{n-1} \oplus x_n)..(x_{k+1} \oplus x_{k+2})\bar{x}_{k+1}1\ 0\ 0..0\ 0$$

e:

$$y_i \oplus y_{i+1} = 0\ 0\ ..0\ 1..0\ 0..0\ 0 \qquad (2.9.2)$$

Quindi, quando y_i rappresenta in codice Gray un numero dispari, si ottiene la rappresentazione del numero successivo complementando la prima cifra a sinistra del primo 1 incontrato partendo da destra.

Consideriamo ora un contatore $R = (R_m...R_k...R_0)$ binario riflesso in cui R_k sono gli stati interni di flip-flop T ed indicano il valore delle cifre di rango K in codice Gray.

Chiamiamo con r_k^n l'ingesso del flip-flop T di rango k all'istante n; si avrà:

$$R_k^{n+1} = R_k^n \oplus r_k^n$$

da cui:

$$R_k^{n+1} \oplus R_k^n = r_k^n$$

Conoscendo lo stato di parità del numero R^n, si possono dedurre dalla (2.9.1) e (2.9.2), le r_k^n, cioè i valori di ingresso da dare ai singoli flip-flop in modo da ottenere il numero R^{n+1}.

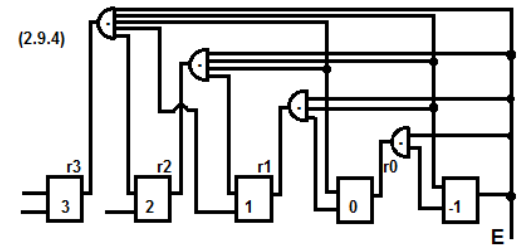
Ora, la parità del numero R^n consideerato è dato dal digit B_0 della rappresentazione in binario puro.

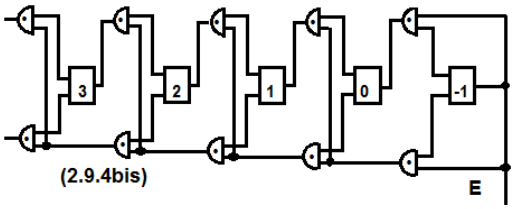
$$B_0 = \oplus \sum_{k=0}^{k} = mR_k$$
$$B_0 = 0\ per\ R^n\ pari$$
$$B_0 = 1\ per\ R^n\ dispari$$

Si deducono le seguenti relazioni:

$$r_0^n = R_0^n \oplus R_0^{n+1} = \bar{B}_0^n E^n \qquad (k=0)$$
$$r_k^n = R_k^n \oplus R_k^{n+1} = R_{k-1}^n \bar{R}_{k-2}^n ... \bar{R}_1^n \ \bar{R}_0^n \ B_0^n \ E^n \quad (1 = k = m)$$

dove E rappresenta la variabile d'ingresso del contatore.





Ponendo:

$$R_{-1}^n = \bar{B}_0^n$$
$$R_{-1}^n = 0 R^n \text{ è dispari}$$
$$R_{-1}^n = 1, R^n \text{è pari}$$

Si ha allora per $0 \leq k \leq m$

$$r_k^n = R_k^n \oplus R_k^{n+1} = R_{k-1}^n \bar{R}_{k-2}^n \bar{R}_{k-3}^n ... \bar{R}_1^n \bar{R}_0^n \bar{R}_{-1}^n E^n \quad (2.9.3)$$

$$\bar{R}_{-1}^n = \oplus \sum_{k=0}^{m} R_k^n$$

Nelle figure (2.9.4 e 2.9.4bis) sono rappresentati 2 contatori in codice binario riflesso ottenuti applicando le formule (2.9.3) o considerando una decomposizione analoga a quella fatta nel caso dei contatori in binario puro. È da notare inoltre il segno $-$ su tutte le R_i salvo quella di rango $k-1$ e la presenza della variabile R_{-1} rappresentata da un flip-flop supplementare di parità.

Questo flip-flop di rango -1, commuterà ogni volta che si applicherà un 1 al suo ingresso.

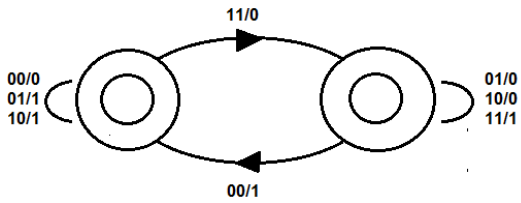
10.10 Diagrammi di stato

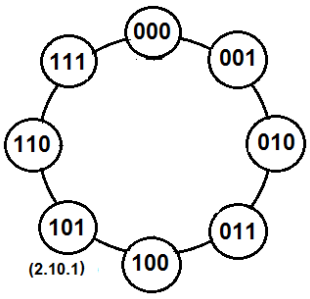
Consideriamo a titolo di esempio, un addizionatore binario seriale (fig.2.1.3) ed assumiamo che la configurazione della memoria M, che conserva fino all'istante $i+1$ il valore del riporto r_{i+1} ottenuto all'istante I, sia rappresentativa dello stato interno del circuito.

r_i \ $x_i\ y_i$	00	01	10	11
0	0/0	0/1	0/1	1/0
1	0/1	1/0	1/0	1/1

diagramma di stato

A partire da uno stato iniziale $r_i = 0$ risulta che le tre combinazioni di ingresso di x_i y_i (cfr. 2.1.2) 00,01,10 il valore di r_i, e cioè lo stato interno del circuito non cambia; mentre per la combinazione $x_i = y_i = 1$ lo stato interno passa dal valore 0 al valore 1. Analogamente si può considerare per quali combinazioni delle variabili d'ingresso lo stato interno rimane uguale ad 1 e per quale valore cambia invece nuovamente da 1 a 0. Si può quindi riscrivere la tavola della verità di 2.1.2 sotto la forma di diagramma di stato, in cui x_iy_i rappresenta la combinazione dei valori d'ingresso, r_i lo stato interno; ed in funzione dei valori di r_i e di x_iy_i sono riportati in ogni casella, separati da una /, il nuovo stato interno r_{i+1} ed il valore di uscita s_i.





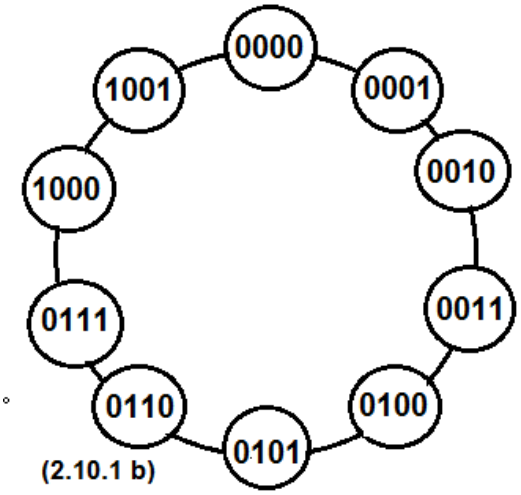
Si può equivalentemente rappresentare il comportamento del circuito per mezzo di un grafo composto da vertici rappresentativi dello stato interno e da archi orientati tra le coppie di vertici.

Gli archi orientati sono rappresentativi dei valori di ingresso che determinano il passaggio dello stato interno dasl vertice di partenza a quello di arrivo, e recano, separati da una barra, oltre a questi valori, anche i corrispondenti valori di uscita.

Nell'esempio in esame si ha quindi il grafo posto di fianco:

Il grafo di una rete sequenziale contiene dunque le stesse informazioni della tabella della verità; con esso però si mette in particolare evidenza la connessione logica degli stati che il sistema può assumere. Una altra rappresentazione molto usata ha nome di diagramma di stato ed indica gli stati successivamente assunti dal sistema.

Un semplice esempio riguarda il diagramma di stato di un contatore binario modulo 2^3 in (2.10.1 a). Il nujmero binario racchiuso nel circolo indica lo stato dei tre flip-flop, e le frecce indicano la sequenza dei cambiamenti di stato (cnf: 2.7).



(2.10.1 b)

Un contatore decimale, o decade, richieder invece 10 stati distinti. Il diagramma di stato. Il diagramma di stato di un tale contatore con codice 8.4.2.1 è riportato in (2.10.1 b).

Se come visto nel 2.7.3 con a_{it} (i=1,2,3,4) si rappresentano i valori di input capaci di cambiare lo stato dei quattro rispettivi flip-flop, dal diagramma di statosi possono ottenere, senza costruire la tabella della verità, le equazioni di input ricavate nel 2.8. Ad esempio, infatti, considerando che i numeri racchiusinel circoletto rappresentano i valori assunti di volta in volta dall'insieme ordinato $A_4\ A_3\ A_2\ A_1$ e se consideriamo in particolare lo stato A_2, inizialmente posty a 0, notiamo che detto stato commuta sempre e solo a partire da una configurazione del tipo $0\ -\ -1$ e quindi l'equazione di input relativa al flip-flop di stato A_2 tenuto conto della variabile d'ingesso p, deve essere:

$$a_{2t} = A_1\ A_2\ p$$

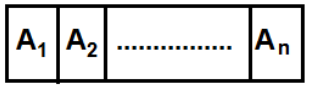
11

Circuiti di un calcolatore digitale (b)

11.1 Registri di scorrimento

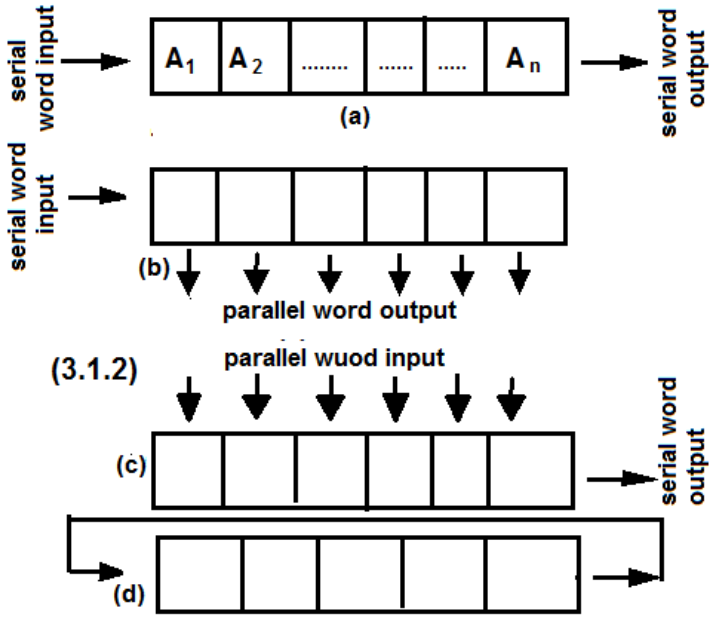
Un registro in un calcolatore digitale è un circuito di memorizzazione di ampiezza, in generale, pari a una parola (o voce) di macchina.

Poiché una parola di macchina consiste di un numero determinato di **bit**, e poiché un flip-flop è un elemento di memorizzazione binario, un registro è costituito da un insieme di flip-flop.



(3.1.1)

In figura 3.1.1 viene data la rappresentazione simbolica di un registro: ogni quadrato rappresenta un flip-flop.



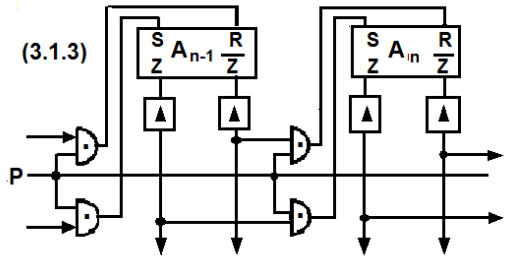
La lettera a indica il registro a e la lettera A_i il flip-flop i-esimo.

Un registro a scorrimento (o shift rergister) permette di fare scorrere il proprio contenuto sulla destra, o sulla sinistra o in entrambe le direzioni; in fig. 3.1.2a compare la rappresentazione simbolica di un registro di shift. Un registro di shift a **n bits** fa scorrere un bit alla volta richiedendo quindi n operazioni di shift per inserire o estrarre una parola seriale di **n bits**.

Circuiti addizionali sono richiesti nel registro di shift se una parola deve essere inserita o prelevata in parallelo: infatti un registro può convertire una parola seriale in una parallelo (3.1.2b) o una parallela in una seriale (3.1.2c).

Connettendo l'output all'input di un registro di shift, quest'ultimo un "'circulating register (3.1.2d)

11.1.1 Equazioni logiche di un registro di shift



Un registro di shift richiede una memorizzazione temporanea tra due flip-flop adiacenti in modo che venga conservata l'informazione di un flip-flop che sta per cambiare e utilizzare tale memorizzazione come input per il flip-flop successivo. In fig. 3.1.3 compare un registro di shift in cui l'elemento base è costituito da flip-flop SR.

L'output di questo flip-flop SR viene indicato con lettera maiuscola A_n o il complemento $\bar{A}_n$; gli input sono indicati con lettere minuscole, a_{nr} e a_{ns}, e rappresentano gli input **reset** e **set** del n-simo flip-flop.

La lettera p rappresenta il segnale di clock che determina l'operazione di **shift**. Come si è già visto a proposito dei contatori, anche per i registri di shift vi sono due tipi di equazioni logiche.

p	$A_{n-1}(t)$	$A_n(t+\tau)$
1	0	0
1	1	1

Il primo tipo descrive lo stato del registro : gli stati dei singoli flip-flop di un registro a scorrimento sono rappresentati nella tabella affiancata, da cui si deduce l'equazione di stato

$$A_n(t) = A_{n-1} \cdot p$$

Questa equazione indica che lo stato dello n-esimo flip-flop dopo l'intervallo temporale τ è lo stato del flip-flop (n-1)-esimo.

P	$A_{n-1}(t)$	$A_n(t+\tau)$	A_{nr}	A_{ns}
1	0	0	1	0
1	1	1	0	1

Il secondo tipo di equazioni logiche descrive l'input del flip-flop. I valori che devono essere assunti dagli impulsi **Set** e **Rerset** del flip-flop n-desimo sono rappresentati nella figura adiacente.

Da cui si deducono le equazioni di input dei flip-flop di un registro di shift:

$$a_{nr} = \bar{A}_{n-1} \cdot p$$
$$a_{ns} = A_{n-1} \cdot p$$

Da queste considerazioni è stato dedotto il circuito di fig. 3.1.3

11.2 Trasferimento dell'informazione

L'informazione conservata in un registro costituisce una parola: mediante un trasferimento di informazione, cerchiamo di realizzare il trasferimento della una parola da un registro ad un altro o un trasferimento multiplo da uno o più registri ad altri registri.

L'importanza di una operazione di trasferimento consiste non solo nel trasferimento stesso, ma nella possibilità di compiere operazioni logiche sulla parola durante il trasferimento. Il trasferimento dell'informazione tra vari registri è l'operazione base mediante la quale possono essere realizzati la elaborazione dei dati, le decisioni logiche, ecc., in un calcolatore: inizialmente le informazioni sono conservate in una unità di memoria da cui vengono trasferite nei vari registri secondo una certa sequenza.

Sono stati sviluppati dei metodi simbolici per poter rappresentare questo processo di trasferimento delle informazioni: in primo luogo un trasferimento viene espresso mediante una istruzione simbolica.

Le equazioni logiche e i diagrammi logici possono essere quindi ottenuti dalle istruzioni simboliche.

Per poter descrivere un tale procedimento simbolico si sono mostrati nel seguente elenco i simboli utilizzati.

Simboli	Descrizione dei simboli
Lettere maiuscole	indicano i registri
Indice i	Denota lo i^{esimo} bit di un registro
Parentesi ()	Indicano il contenuto di un registro
Doppia freccia $\Longrightarrow$	Indica il trasferimento di una parola da un registro all'altro
Parentesi []	Porzione di registro il cui contenuto ha una dipendenza funzionale del registro stesso
Due punti :	Due punti (preceduti da una variabile booleana) indicano il verificarsi dell'istruzione seguente quando la variabile ha il valore 1
Freccia semplice $\longrightarrow$	Indica una sequenza di stati
Segno +	Questo simbolo ha due significati: 1) una operazione logica OR quando esso è usato tra variabili booleane; 2) un'addizione aritmetica quando è usato tra numeri
Segno -	Questo simbolo indica una operazione aritmetica

Per esempio (A) indica il contenuto del registro (A) e (A_i) indica il contenuto dell'i^{esimo} bit del registro A; (A,B) indica il contenuto dei registri A e B che vengono usati insieme come un unico registro.

Con (A)$\Longrightarrow B$ si indica che il contenuto del registro A viene trasferito in B e $0 \Longrightarrow A$ significa che degli zeri vengono inseriti in A. L'operazione di shift a sinistra viene indicata con $(A_i) \Longrightarrow A_{i+1}$, supponendo che l'indice i incrementato indichi un bit significativo di rango superiore.

L'operazione di shift a destra sarà allora:

$$(A_i) \Longrightarrow A_{i-1}$$

Tabella 11.1: Tabella A

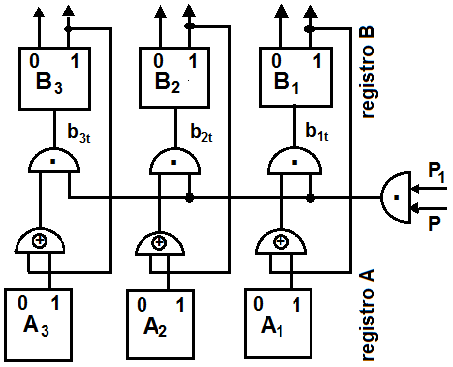
p	A_1	B_1	B_{it}
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Le espressioni simboliche $S_n[A]$ e $N_m[A]$, indicano la parte segno e la parte numerica del registro A. L'espressione $P_i : 0 \Longrightarrow A$ indica che, quando p_i ha il valore 1, viene eseguita l'istruzione che azzera il registro A.

Mediante questo simbolismo, si può descrivere una operazione di trasferimento: supponiamo di avere una parole di 3 bit nel registro A e di trasferirla in B al momento in cui viene dato il via da un segnale P_1 proveniente dall'unità di controllo del calcolatore.

Tale operazione si esprimerà nella forma

$$P_1 : (A) \Longrightarrow B \qquad (3.2.2)$$



Supponendo che i registri siano costituiti da flip-flop T, dalla (3.2.2) si deduce la tavola della verità (3.2.3).

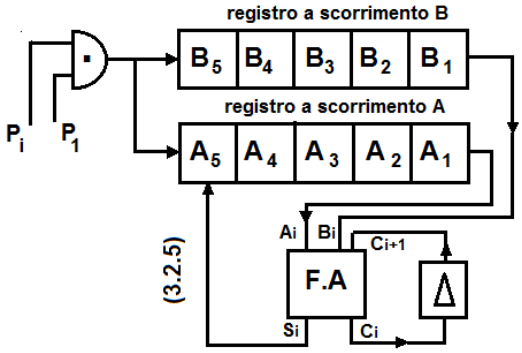
In questa tabella, il trasferimento inizia quando l'impulso p dell'orologio è uguale ad 1: b_{it} indica il valore d'ingresso del flipo-flop rappresentativo dell'i^{esimo} bit nel registro B.

$$b_{it} = (A_i \oplus B_i) \cdot P_1 \cdot P \qquad per \quad i = 1, 2, 3$$

Queste equazioni indicano che l'0perazione di trasferimento in un registro B costituito da flip-flop T viene realizzato mediante un XOR. Nella figura collocata di fianco compare il diagramma logico (trasferimento in parallelo di una parola dal registro A al registro B) che descrive la connessione dei due registri.

L'esempio presente si riferisce ad un trasferimento in parallelo di una parola: una informazione può essere però trasferita in forma sequenziale.

Come esempio, consideriamo la somma aritmetica tra due numeri binari entrambi di cinque bit: siano essi conservati nei registri di shift A e B (fig. 3.2.5). L'addizione avviene quando P_1, proveniente dalla unità di controllo, assume il valore 1.



Nella addizione seriale, coppie di bits corrispondenti dei due numeri vengono sommati allo stesso istante, quindi i due numeri vengono fatti scorrere del registro a trasferimento dagli impulsi di orologio $P_1, P_2, P_3, P_4 e P_5$.

Il trasferimento può essere espresso dalla seguente formula simbolica:

$$\begin{aligned} P_1P_i: \quad & (B_i) \Longrightarrow B_{i-1} \\ & 0 \Longrightarrow B_5 \\ (A_i) \Longrightarrow A_{i-1} \quad & per\ i = 1, ..., 5 \\ & S_i \Longrightarrow A_5 \\ & C_{i+1} = \triangle C_i \end{aligned}$$

dove S_i e C_i sono le funzioni descritte nel cap. II parte II.

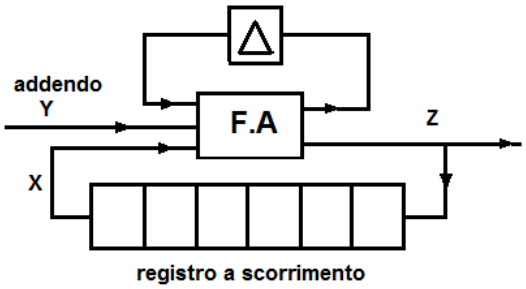
Le prime tre istruzioni simboliche indicano lo shift a destra dei registri A e B.

Le ultime due istruzioni descrivono la connessione del bit di riporto e la memorizzazione del bit somma nel flip-flop A_5.

Il diagramma logico di questo addizionatore binario seriale è mostrato in (3.2.5)

Il metodo simbolico visto ora può essere utilizzato sia per costruire i diagrammi logici di un circuito, sia le equazioni logiche di quest'ultimo. Nella figura compare il diagramma logico di un accumulatore binario seriale. Esso consiste in un addizionatore seriale per eseguire la somma e in un registro a scorrimento per conservare uno degli addendi prima della addizione o la somma dopo l'addizione.

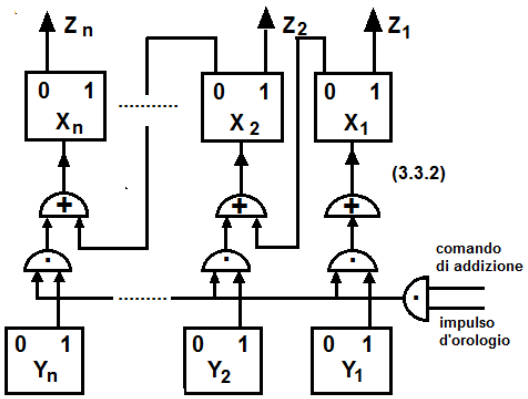
11.3 Accumulatori binari



Per accumulatore binario, si intende un circuito in cui un registro memorizza un numero binario e appena ne riceve un'altro, quest'ultimo viene sommato (o sottratto) dal primo numero e quindi viene memorizzata la somma (o la differenza). Un accumulatore può essere costruito per operazioni in parallelo o in serie.

Nella figura compare il diagramma logico di un accumulatore binario seriale. Esso consiste in un addizionatore seriale per eseguire la somma e in un registro a scorrimento per conservare uno degli addendi prima della della addizione o la somma dopo l'addizione.

La sottrazione può essere ottenuta mediante la somma del complemento a 2 del numero.



Se l'addizionatore seriale di fig. 3.3.1 viene sostituito con un sottrattore seriale o con un addizionatore-sottrattore seriale, l'accumulatore può eseguire la sottrazione o entrambe le operazioni di somma e sottrazione.

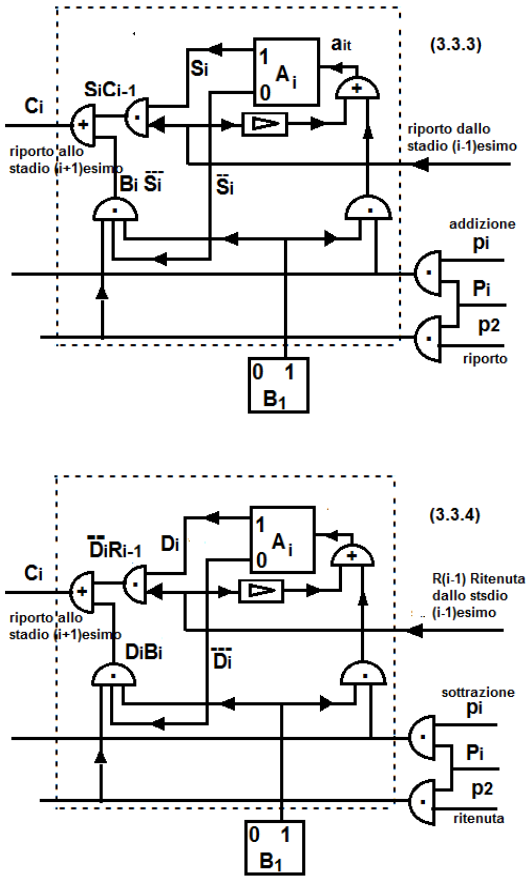
In figura 3.3.2 viene mostrato un accumulatore binario parallelo.

Ogni flip-flop dell'accumulatore funziona come un contatore modulo 2.

Il primo addendo è inizialmente conservato in questi contatori

Durante l'addizione ogni contatore contenga gli impulsi rappresentativi del bit di addendo o di riporto, e genera un segnale di riporto quando il suo stato cambia da 1 a 0.

Il diagramma logico di un accumulatore binario parallelo relativo all'addizione viene ora descritto mediante il metodo simbolico.



In fig. 3.3.3 consideriamo l'i-esimo stadio: l'addizione può essere decomposta in due passi; durante il primo vengono sommati i bits A_i e B_i, durante il secondo vengono addizionate la somma risultante dal primo passo e il bit di riporto dovuto all'operazione di somma compiuta sui bit di rango inferiore.

Le istruzioni simboliche di questi due passi sono:

$$P_1 p_1 : B_i \cdot (\bar{A}_i) + \bar{B}_i(A_i) \Longrightarrow A_i$$

$$P_1 p_1 : \; C_{i-1}(\bar{A}_i) + \bar{C}_{i-1} \; (A_i) \Longrightarrow A_i$$

In queste istruzioni la somma di A_i e B_i avviene al clock pulse p_1; la somma del risultante bit A_i e del riporto C_{i-1} avviene all'impulso seguente p_2. Entrambe le addizioni hanno luogo durante lo stato P_1 che da l'ordine di addizionare da parte dell'unità di controllo.

Il bit di riporto può essere formato in più modi.

Per esempio:

$$C_i = B_i \bar{S}_i + C_{i-1} S_i$$

dove S_i e $\bar{S}_i$ sono le uscite del flip-flop A_i dopo il clock pulse P_1

La linea di ritardo assicura che il cambio di stato dello flip-flop A_i avvenga dopo che si sia formato il nuovo riporto.

Un accumulatore può essere anche predisposto ad eseguire una sottrazione. In questo caso le istruzioni simboliche relative alla differenza sono le stesse di quelle relative alla somma mentre cambia l'espressione booleana che dà il riporto.

Chiameremo inoltre P_2 lo stato del calcolatore che indica l'operazione di sottrazione.

$$P_2p_1 : B_i(\bar{A}_i) + \bar{B}_1(A_i) \Longrightarrow A_i$$
$$P_2p_2 : R_{i-1}(\bar{A}_i) + \bar{R}_{i-1}(A_i) \Longrightarrow A_i$$

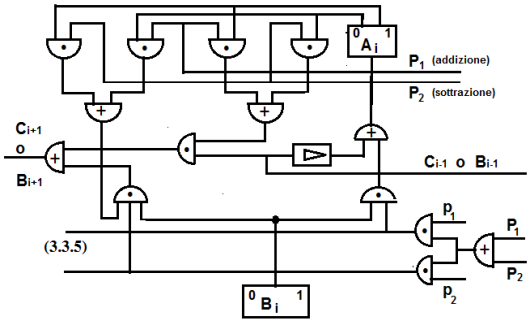
dove R_i è la ritenuta:

$$R_i = D_iB_i + \bar{D}_iR_{i-1}$$

essendo D_i la differenza allo i$^{\text{simo}}$ stadio.

Nella figura 3.3.4 compare il diagramma logico di un accumulatore-sottrattore.

Nella figura 3.3.4 compare il diagramma logico di un accumulatore-sottrattore.



Si può ora formare un accumulatore relativo sia all'addizione che alla sottrazione.

Esaminando infatti i diagrammi 3.3.3 e 3.3.4 si vede che l'unica differenza tra i due consiste nella connessione della uscita del flip-flop A_i.

Un accumulatore per addizione e sottrazione può allora essere costruito usando quattro addizionali porte di AND e due addizionali circuiti OR. (fig-3.3.5)

Quando lo stato è P_1, l'accumulatore somma, quando lo stato è P_2, avviene la sottrazione.

11.4 Matrici di commutazione

Un importante circuito di un elaboratore digitale è costituito dalle matrici di commutazione, descritte nel seguito del paragrafo mediante l'ausilio delle matrici booleane.

11.4.1 Circuiti di commutazione con uscita multipla

Un circuito di commutazione con output multiplo può essere descritto mediante un certo numero di funzioni booleane.

Prendiamo come esempio il full-adder relativo ad un singolo bit.

Le tavole della verità sono date nel capitolo II parte II e da queste si ottengono le formule per la somma S e il riporto C_0:

$$S = \bar{X}$$
$$\bar{Y}C_i + \bar{X}Y\bar{C}_i + x\bar{Y}\bar{C}_i + XYC_i$$
$$C_0 = \bar{X}YC_i + X\bar{Y}C_i + XY\bar{C}_i + XYC_i$$

dove X, Y e C_i sono rispettivamente gli addendi, e il bit di riporto.

Queste funzioni possono essere scritte mediante i prodotti fondamentali:

$$S = m_1 + m_2 + m_4 + m_7$$

$$C_0 m_3 + m_5 + m_6 + m_7$$

Le due funzioni S e C_0 possono però essere espresse anche come:

$$S = (0\times m_0)+(1\times m_1)+(1\times m_2)+(0\times m_3)+(1\times m_4)+(0\times m_5)+(0\times m_6)+(1\times m_7) \quad (3.4.1)$$

$$C_0 = (0\times m_0)+(0\times m_1)+(0\times m_2)+(1\times m_3)+(0\times m_4)+(1\times m_5)+(1\times m_6)+(1\times m_7) \quad (3.4.2)$$

Queste due ultime equazioni possono essere rappresentate mediante matrici booleane, come segue:

$$\begin{bmatrix} S \\ C \end{bmatrix} = \begin{bmatrix} 0\,1\,1\,0\,1\,0\,0\,1 \\ 0\,0\,0\,1\,0\,1\,1\,1 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \end{bmatrix}$$

(3.4.3)

o con notazione matriciale:

$$\Big[0\Big] = \Big[B\Big] \times \Big[m\Big]$$

dove [0], [B] e [m] sono rispettivamente la matrice di uscita, la matrice booleana B e la matrice dei prodotti fondamentali.

La moltiplicazione della matrice B per la matrice m segue le regole ordinarie della moltiplicazione dell'algebra matriciale.

In altre parole, gli elementi della matrice 0 sono la somma dei prodotti dei termini di m per i corrispondenti bit della matrice B, come mostrato nella 3.4.1 e 3.4.2.

La matrice B è costituita da due righe e otto colonne: ogni riga rappresenta una uscita mentre ogni colonna rappresenta un **minterm**.

Gli 1 della riga 01101001 rappresentano i termini $m_1, m_2, m_4 m_7$ mentre quelli della riga 00010111 rappresentano i termini m_3, m_5, m_6, m_7: cioè la matrice booleana B rappresenta un insieme dei vettori caratteristici delle finzioni booleane S e C_0.

Le espressioni booleane per S e C_0 possono essere scritte mediante i **maxterm** (somme fondamentali).

$$S = M_0\ M_3\ M_5\ M_6$$

$$C_0 = M_0\ M_1\ M_2\ M_4$$

espressioni equivalenti a:

$$S = (0+M_0)(1+M_1)(1+M_2)(0+M_3)(1+M_4)(0+M_5)(0+M_6)(1+M_7) \quad (3.4.4)$$

$$C_0 = (0+M_0)(0+M_1)(0+M_2)(1+M_3)(0+M_4)(1+M_5)(1+M_6)(1+M_7) \quad (3.4.5)$$

Rappresentando le formule (3.4.3) e (3.4.4) mediante matrici booleane, si ha:

$$\begin{bmatrix} S \\ C_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \\ M_7 \end{bmatrix} \quad (3.4.6)$$

o con la notazione matriciale:

$$\Big[0\Big] = \Big[B\Big] \times \Big[M\Big]$$

dove [M] è la matrice M dei maxterm.

In questo caso la moltiplicazione della matrice B per la matrice M non segue le regole ordinarie della moltiplicazione dell'algebra matriciale.

Infatti, come si vede dalle (3.4.4) e (3.4.5) il generico elemento della matrice prodotto è definito come il prodotto delle somme dei singoli termini di M con il corrispondente bit della matrice B.

Le funzioni (3.4.3) e (3.4.6) ottenute mediante notazioni matriciali, risultano espresse nelle due forme canoniche e le matrici booleane B delle due forme risultano identiche.

Quindi, quando di un circuito a più uscite, si conosce la matrice B della notazione matriciale booleana e quest'ultima è data in forma canonica, sono conosciute immediatamente anche le equazioni booleane di tutti gli output, sia nella forma canonica disgiuntiva che in quella congiuntiva.

Ovviamente si possono trovare immediatamente le espressioni $\bar{S}$ e $\bar{C}_0$. Infatti:

$$\bar{S} = m_0 + m_3 + m_5 + m_6$$

$$\bar{C}_0 = m_0 + m_1 + m_2 + m_4$$

oppure

$$\Big[\bar{0}\Big] = \Big[\bar{B}\Big] \times \Big[m\Big]$$

Da cui le espressioni matriciali booleane:

$$\begin{bmatrix} \bar{S} \\ \bar{C}_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \end{bmatrix} \quad (3.4.7)$$

oppure

$$\left[\bar{0}\right] = \left[\bar{B}\right] \times \left[m\right]$$

dove $[\bar{0}]$ è la matrice di uscita complementare e $[\bar{B}]$ è la matrice B complementata.

È da notare che ogni bit di $\bar{B}$ è il complemento a 1 del corrispondente bit della matrice B.

La moltiplicazione della matrice $\bar{B}$ per la matrice m segue le regole della moltiplicazione dell'algebra matriciale.

Usando i maxterms, si ha invece:

$$\bar{S} = M_1\ M_2\ M_4\ M_7$$
$$\bar{C}_0 = M_3\ M_5\ M_6\ M_7$$

da cui, le forme matriciali sono:

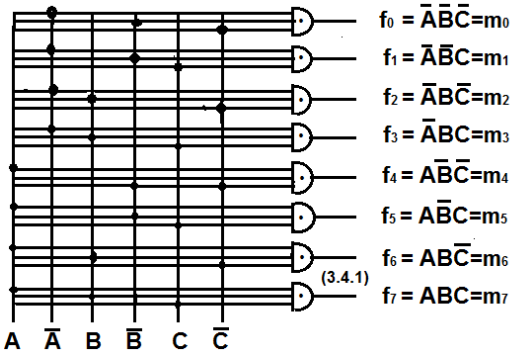
$$\begin{bmatrix} \bar{S} \\ \bar{C}_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \\ M_7 \end{bmatrix} \qquad (3.4.8)$$

oppure:

$$\left[\bar{0}\right] = \left[\bar{B}\right] \times \left[M\right]$$

La moltiplicazione della matrice B con la matrice M non segue le regole ordinarie dell'algebra matriciale ma è definita da (3.4.4) e (3.4.5).

11.4.2 Matrici di commutazione rettangolari



La matrice di commutazione è un circuito con output multiplo: essa realizza una matrice booleana.

Le matrici più comuni sono quelle rettangolari, ad albero e a doppio albero.

Esistono però anche altri tipi di matrici.

Una matrice rettangolare che utilizza solo porte di AND è rappresentata in fig. 3.4.1.

Tale matrice è relativa a tre variabili booleane e coinvolge nella sua realizzazione tre coppie di input e otto output.

Per ogni prodotto delle tre variabili di input esiste uno e uno solo output.

Le funzioni booleane dei 8 output sono:

$$f_0 = m_0 = \bar{A}\bar{B}\bar{C} \qquad f_1 = m_1 = \bar{A}\bar{B}\ C$$

$$f_2 = m_2 = \bar{A}\ B\bar{C} \qquad f_3 = m_3 = \bar{A}\ B\ C$$

$$f_4 = m_4 = A\bar{B}\bar{C} \qquad f_5 = m_5 = A\bar{B}\ C$$

$$f_6 = m_6 = A\ B\bar{C} \qquad f_7 = m_7 = A\ B\ C$$

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix} = \begin{bmatrix} 10000000 \\ 01000000 \\ 00100000 \\ 00010000 \\ 00001000 \\ 00000100 \\ 00000010 \\ 00000001 \end{bmatrix} \times \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \end{bmatrix}$$

Il circuito può anche essere espresso mediante una forma matriciale:

In questa matrice booleana compare un solo 1 in ogni riga e tutti gli 1 si vengono a trovare sulla diagonale principale.

Per la realizzazione del circuito non sono necessarie porte di OR. Questa è la forma più semplice che può assumere una matrice B relativa ad una matrice di commutazione avente tre coppie di input ed otto output.

Si vede da fig. 3.4.1, che per tre variabili sono necessarie otto porte di AND a tre ingressi. All'input di ogni AND led tre variabili di input possono comparire sia in forma diretta che in forma complimentata.

Per n coppie di variabili di input, il numero di input relativi alla porta di AND diventa di $n2_n$; cioè un numero molto elevato per un valore grande di n.

Matrici più economiche si ottengono mediante strutture ad albero 0 a doppio albero (vedi fig. 3.4.3).

Una matrice rettangolare con tre variabili di input e utilizzante porte di OR risulta essere uguale a quella di fig. 3.4.1, dove le porte di AND sono sostituite con porte di OR.

Le funzioni booleane delle otto uscite sono:

$$f_0 = M_0 = A + B + C \qquad f_1 = M_1 = A + B + \bar{C}$$

$$f_2 = M_2 = A + \bar{B} + C \qquad f_3 = M_3 = A + \bar{B} + \bar{C}$$

$$f_4 = M_4 = \bar{A} + B + C \qquad f_5 = M_5 = \bar{A} + B + \bar{C}$$

$$f_6 = M_6 = \bar{A} + \bar{B} + C \qquad f_7 = M_7 = \bar{A} + \bar{B} + \bar{C}$$

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix} = \begin{bmatrix} 01111111 \\ 10111111 \\ 11011111 \\ 11101111 \\ 11110111 \\ 11111011 \\ 11111101 \\ 11111110 \end{bmatrix} \times \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \\ M_7 \end{bmatrix}$$

La matrice di commutazione si può allora rappresentare nella forma logica:
Il numero di input per le porte OR risulta essere anche esso 2^n

11.4.3 Matrici di commutazione ad albero

In figura 3.4.2 è rappresentata una matrice di commutazione ad albero costruita da circuiti AND e con quattro variabili booleane per input.

Vi sono quindi quattro coppie di input e 16 output.

Le equazioni booleane dei 16 output sono:

$$f_0 = [(\bar{A}\bar{B})\bar{C}]\bar{D} \quad f_5 = [(\bar{A}B)\bar{C}]\bar{D} \quad f_{10} = [(A\bar{B})C]\bar{D}$$

$$f_1 = [(\bar{A}\bar{B})\bar{C}]D \quad f_6 = [(\bar{A}B)C]\bar{D} \quad f_{11} = [(A\bar{B})C]D$$

$$f_2 = [(\bar{A}\bar{B})C]\bar{D} \quad f_7 = [(\bar{A}B)C]D \quad f_{12} = [(AB)\bar{C}]\bar{D} \quad (3.4.9)$$

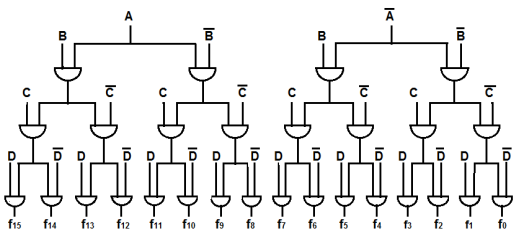
$$f_3 = [(\bar{A}\bar{B})C]D \quad f_8 = [(A\bar{B})\bar{C}]\bar{D} \quad f_{13} = [(AB)\bar{C}]D$$

$$f_4 = [(\bar{A}B)\bar{C}]\bar{D} \quad f_9 = [(A\bar{B})\bar{C}]D \quad f_{14} = [(AB)C]\bar{D} \quad f_{15} = [(AB)C]D$$

Le precedenti equazioni booleane possono essere scritte in forma matriciale. In questo caso, la matrice booleana è a livello multiplo ed i diversi livelli vengono indicati dalle precedenti parentesi tonde e quadre.

Vedremo in seguito tali rappresentazioni matriciali.

La seguente (3.4.2) è la rappresentazione di una matrice di commutazione ad albero con circuiti AND.



Nelle equazioni (3.4.9), le parentesi tonde e quadre indicano anche la maniera di connettere i blocchi di AND di fig. 3.4.2.

Per $n = 2$, sono necessari 2_3 input per gli AND-

Per $n = 3$, sono richiesti per gli AND $2_3 + 2_4$ inputs.

Per **n coppie** di variabili, è richiesto un numero N di inputs per ogni AND, dove N è dato da:

$$N = \sum_{r=2}^{n} 2^{r+1}$$

Rispetto alla matrice rettangolare, la matrice ad albero richiede un minor numero di inputs di AND.

Funzioni booleane di matrici di matrici ad albero aventi blocchi di OR hanno espressioni simili a quelli di (3.4.9), in termini però di somme fondamentali e la loro realizzazione circuitale si ottiene da quella di fig. 3.4.2 sostituendo alle porte di AND quelle di OR.

In fig. 3.4.3 viene mostrata una matrice a doppio albero con porte di AND, quattro coppie di variabili di input e 2^4 outputs.

Questa configurazione richiede un numero di input di AND inferiore a quello delle configurazioni ad albero o rettangolari.

Le equazioni booleane dei 16 output del circuito sono:

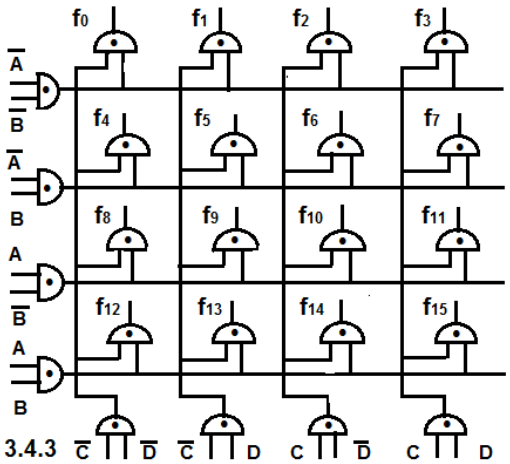
$$f_0 = (\bar{A}\bar{B})(\bar{C}\bar{D}) \quad f_5 = (\bar{A}B)(\bar{C}D) \quad f_{10} = (A\bar{B})(C\bar{D})$$

$$f_1 = (\bar{A}\bar{B})(\bar{C}D) \quad f_6 = (\bar{A}B)(c\bar{D}) \quad f_{11} = (A\bar{B})(CD)$$

$$f_2 = (\bar{A}\bar{B})(C\bar{D}) \quad f_7 = (\bar{A}B)(CD) \quad f_{12} = (AB)(\bar{C}\bar{D}) \quad (3.4.10)$$

$$f_3 = (\bar{A}\bar{B})(CD) \quad f_8 = (A\bar{B})(\bar{C}\bar{D}) \quad f_{13} = (AB)(\bar{C}D)$$

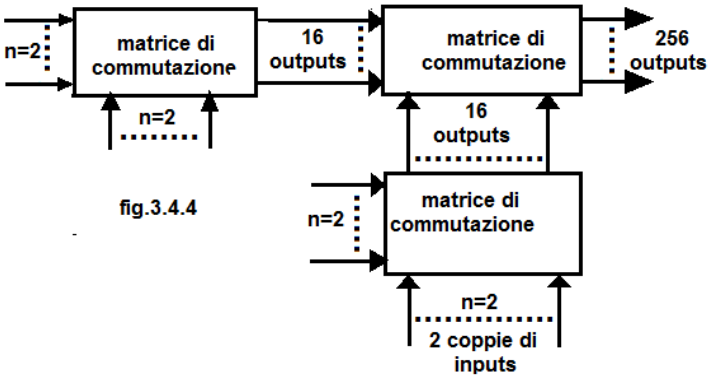
$$f_4 = (\bar{A}B)(\bar{C}\bar{D}) \quad f_9 = (A\bar{B})(\bar{C}D) \quad f_{14} = (AB)(\bar{D}) \quad f_{15} = (AB)(CD)$$



3.4.3

Le funzioni precedenti possono essere scritte mediante una formulazione matriciale: la matrice booleana è anche in questo caso a livelli multipli.

Le parentesi delle equazioni (3.4.10) indicano il modo di connettere i blocchi di AND di fig. 3.4.3.



In figura 3.4.4 viene mostrato come si può realizzare una matrice con otto coppie di input e 256 output.

Gli otto input vengono divisi in due gruppi di quattro variabili ciascuno.

Se n non è multiplo di quattro, si può seguire lo stesso procedimento, solo che la configurazione risultante non sarà simmetrica.

11.4.4 Confronto tra i vari tipi di matrici

coppie di input	output 2	AND input (oppure OR) richiesti		
		rettangolare	ad albero	a doppio albero
2	4	8	8	8
3	8	24	24	24
4	16	64	56	48
5	32	160	120	96
6	64	384	248	176
7	128	896	504	328
8	256	2048	1016	608
9	512	4608	2040	1168
10	1024	10240	4088	2240
11	2048	22528	8184	4368
12	4096	49152	16376	8544

I tre tipi di matrici di commutazione (rettangolare, ad albero, a doppio albero) realizzano le stesse operazioni logiche ma richiedono un numero differente di input per le porte di AND (o di OR).

In tavola 3.4.5 vengono rappresentati i valori assunti dal numero di input per un n compreso tra 2 e 12.

Si noterà il rapido crescere degli input delle porte di AND in una matrice rettangolare, all'aumento di n.

In una matrice ad albero, ogni volta che l'input aumenta di una coppia, il numero richiesto di input diventa più del doppio mentre in una matrice a doppio albero il numero risulta minore del doppio.

Queste differenze diventano molto sensibili quando n è, per es., dell'ordine di 12.

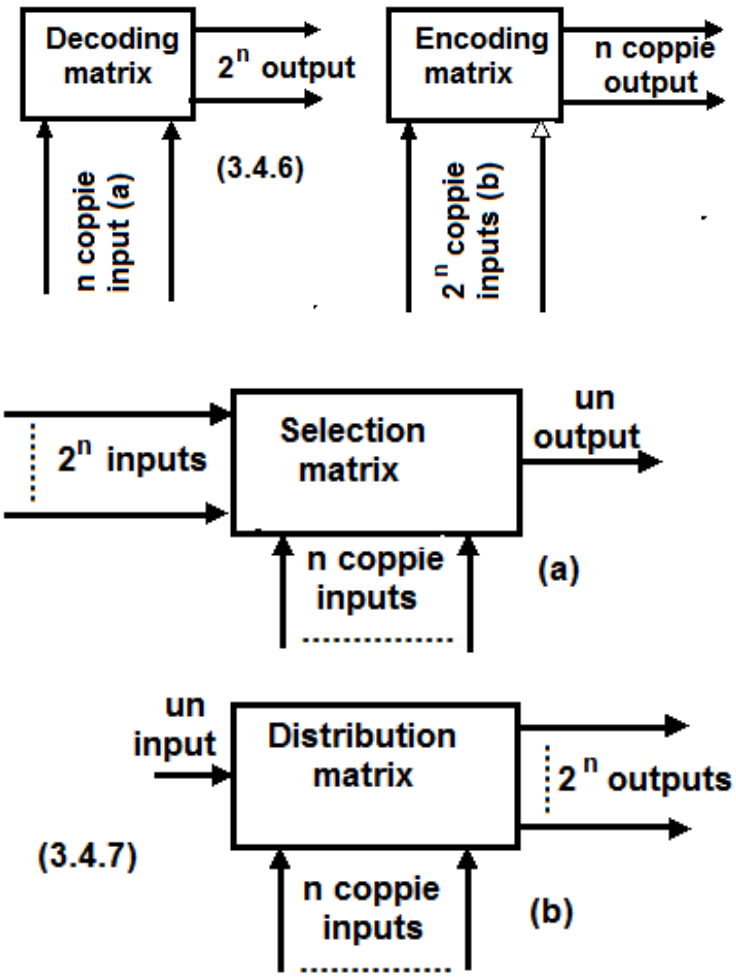
Ma lo studio degli input delle porte di AND non è sufficiente per la scelta della particolare matrice di commutazione.

Un'altra considerazione da fare è quella relativa al numero di livelli che un segnale deve attraversare dall'input all'output.

In una matrice rettangolare, il segnale passa attraverso un solo livello; per una matrice ad albero, esso deve passare attraverso $n - 1$ livelli.

In una matrice a doppio albero il segnale passa attraverso un livello per $n = 2$, due livelli per $n = 3$ e 4, tre livelli per $n = 5$ fino a 8, e quattro livelli per n che va da 9 a 16.

11.4.5 Applicazioni delle matrici di commutazione



In figura 3.4.6 (Matrici di commutazione usate come decodificatori e codificat0ri) e 3.4.7 (Matrici di commutazione usate come selettori e distributori) sono illustrate alcune applicazioni delle matrici di commutazione.

In fig. 3.4.6 (a) appare un decodificatore.

Vi sono n coppie di input e 2^n uscite. Come si vedrà nei capitoli seguenti, una porzione di una parola di istruzione di un calcolatore digitale costituisce un codice operativo ed il decodificatore utilizza tale codice per identificare l'istruzione da seguire e dare inizio alla sua esecuzione.

L'uscita di un decodificatore può essere inferiore a 2^n terminali, se lo si desidera.

Una variante di un decodificatore può essere un traduttore che traduca, per esempio, un numero in codice binario decimale nella corrispondente cifra decimale.

In questo caso vi sono quattro coppie di input (nel caso di un codice a quattro bit) e 10 uscite che rappresentano le 10 cifre decimali.

In fig. 3.4.6(b) viene mostrato un codificatore: in esso vi sono 2^n input e **n coppie** di output.

Per esempio, le ventisei lettere dell'alfabeto possono essere codificate mediante ventisei combinazioni di cinque variabili booleane.

In fig. 3.4.7.(a) viene mostrata una matrice di selezione . In questa matrice vi sono due gruppi di input e una uscita.

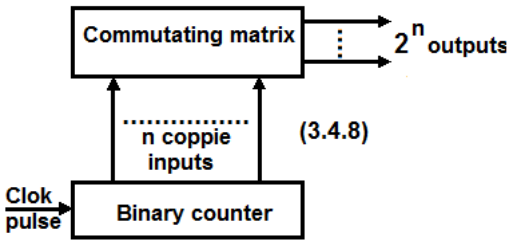
Un gruppo, costituito da 2^n terminali di input, è connesso alla sorgente dei segnali, e l'altro grippo, costituito ds da n coppie di input, viene attivato da un codice selezionato.

In uscita, ad ogni istante, compare un solo segnale. Per esempio, una parola seriale può essere letta dal dispositivo di un canale selezionato su tamburo magnetico.

In questo caso le n coppie di input di fig. 3.4.7(a) rappresentano l'indirizzo del canale, i 2^n input provengono dal dispositivo di lettura e l'output dà la parola letta.

Viceversa, un distributore, come da fig. 3.4.7(b), ha 2^n uscite, un segnale sorgente ed n coppie di input.

Il segnale viene distribuito nelle varie uscite mediante una data combinazione delle n coppie di entrata.



Per esempio, una parola seriale che arriva da una sorgente di segnali viene distribuita al canale selezionato su tamburo magnetico.

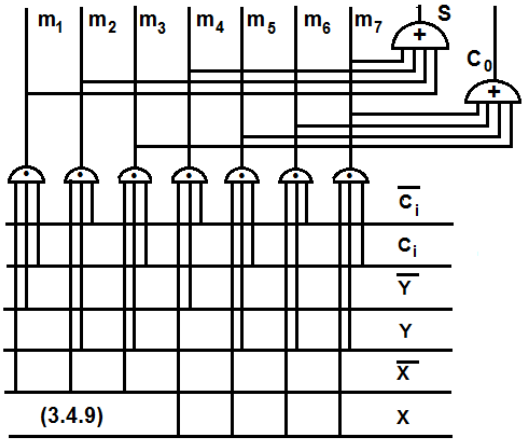
Se le n coppie di fig. 3.4.6(a) provengono da un contatore binario che conteggia una sequenza di impulsi di orologio, il circuito risultante si dice commutatore (fig. 3.4.8):

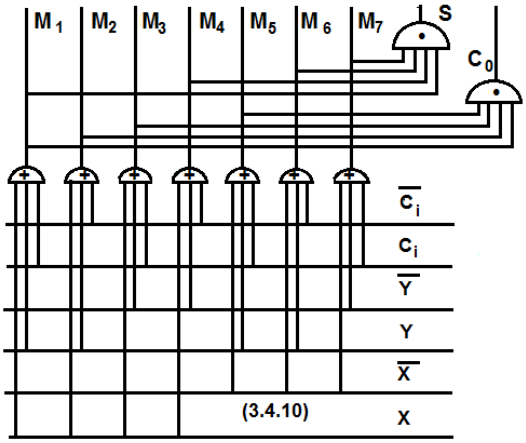
In un commutatore, le uscite sono selezionate una dopo l'altra, secondo la sequenza degli stati del contatore.

Per esempio, un commutatore può essere usato per inviare segnali ai vari canali secondo una particolare sequenza temporale.

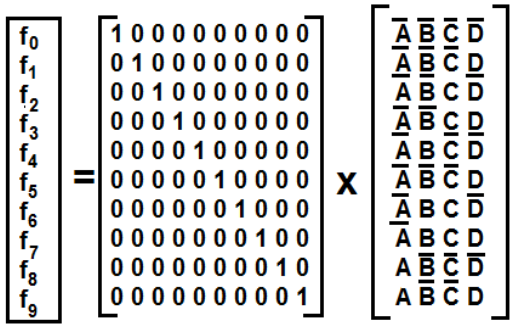
Le matrici possono essere anche usate come generatori di sequenza o come filtri di sequenze.

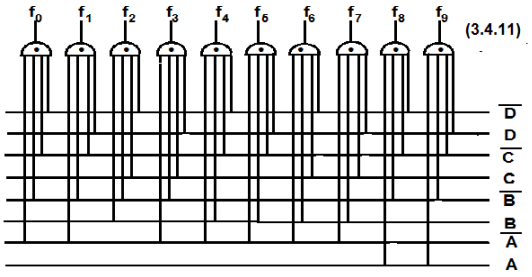
11.4.6 Esempi di matrici di commutazione





8.4.2.1 codid digit A B C D	decimal digit	function
0 0 0 0	0	f_0
0 0 0 1	1	f_1
0 0 1 0	2	f_2
0 0 1 1	3	f_3
0 1 0 0	4	f_4
0 1 0 1	5	f_5
0 1 1 0	6	f_6
0 1 1 1	7	f_7
1 0 0 0	8	f_8
1 0 0 1	9	f_9
1 0 1 0	non usati	
1 0 1 1		
1 1 0 0		
1 1 0 1		
1 1 1 0		
1 1 1 1		





(.a)- Un Full-adder può essere considerato una matrice di commutazione: le equazioni 3.4.2 e 3.4.2, ne danno la rappresentazione booleane.

In fig. 3.4.9 compare il diagramma logico: la porzione di sinistra del circuito è costituito da una matrice rettangolare di porte di AND.

vi sono solo sette porte di AND in quanto nelle equazioni 3.4.1, 3.4.2, compaiono solo sette prodotti fondamnetali (sugli otto possibili determinati dalle tre variabili booleane).

Le linee orizzontali aggiuntive costituiscono gli **input** dei due circuiti OR i cui **output** sono S e C_0: la disposizione delle connessioni degli input per le porte OR corrisponde alla disposizione dei bit 1 e 0 delle righe nella matrice booleana B che compare in 3.4.3 per la rappresentazione matriciale del **Full-adder**.

La matrice rappresentativa di un **Full-adder** viene chiamata una **AND-OR matrix**.

Se la sintesi del circuito viene fatta partendo dalle operazioni 3.4.4, la matrice risultante viene chiamata **OR-AND matrix** (fig.4.4.10)

(.b)- La matrice rettangolare di fig. 3.4.11 è la sintesi della tabella della verità posta qui sul lato destro.

La matrice rappresenta un traduttore che converte un digit codificato nel codice binario 8.4.2.1, nel corrispondente digit decimale.

Poiché delle sedici combinazioni possibili dovute alle quattro variabili booleane solo le prime 10 vengono prese in considerazione, la matrice di fig. 3.4.11, è il diagramma logico delle matrici booleane seguenti

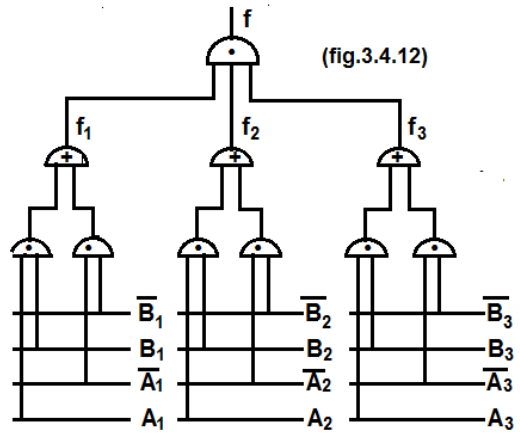
Analogamente si possono costruire matrici di conversione relative ai codici tipo **ad eccesso di tre**. etc.

(.c)- Le matrici **a)** e **b)** sono rappresentate in forma booleana da matrici booleane di tipo elementare.

Volendo usare matrici a più livelli, si possono usare matrici booleane di forma non elementare.

Un circuito di confronto tra due numeri binari è anch'esso un circuito a più uscite: siano A e B le due parole binarie da confrontare. Il modo di operare del

comparatore è descritto dalle seguenti equazioni booleane (supponendo che A e B siano costituite da tre bit ciascuna):



$$\begin{bmatrix} f \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} \times \begin{bmatrix} f_1 & f_2 & f_3 \end{bmatrix}$$

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} 1\,1\,0\,0\,0\,0 \\ 0\,0\,1\,1\,0\,0 \\ 0\,0\,0\,0\,1\,1 \end{bmatrix} \times \begin{bmatrix} \bar{A} & \bar{B} \\ A & B \\ \bar{A} & \bar{B} \\ A & B \\ \bar{A} & \bar{B} \\ A & B \end{bmatrix}$$

$$f = f_1 f_2 f_3$$

$$f_1 = \bar{A}_1 \bar{B}_1 + A_1 B_1$$

$$f_2 = \bar{A}_2 \bar{B}_2 + A_2 B_2$$

$$f_3 = \bar{A}_3 \bar{B}_3 + A_3 B_3$$

Le funzioni f_1, f_2, f_3 diventano rispettivamente uguali ad 1 quando la prima, la seconda e la terza coppia di **bit** corrispondenti sono uguali e la funzione f diventa 1 quando le f_1, f_2, f_3 sono tutte uguali ad 1.

Le quattro funzioni possono essere scritte mediante la forma matriciale non elementare rappresentata a destra

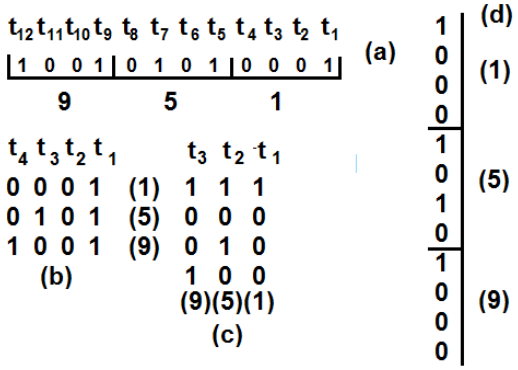
Questa funzione viene realizzata dalla matrice di fig. 3.4.12. Questa è una matrice AND-OR-AND con 21 input. Se la funzione ì viene scritta mediante prodotti fondamentali la corrispondente matrice rettangolare richiede 56 inputs.

11.5 Generatori di sequenze binarie

Per sequenza binaria si intende una sequenza temporale di bit 1 e 0. L'intervallo di tempo che intercorre tra due bit adiascenti viene chiamato **bit time** o **digit time**.

Un numero decimale codificato in BCD può essere rappresentato con diverse sequenze binarie. Per esempio, consideriamo il numero decimale 159. Nel codice 8.4.2.1 esso diventa 001,0101,10001.

In fig. 3.5.1 vengono fatti vedere quattro possibili modi di rappresentare questo numero.



In fig. 3.5.1(a), tutti i digits e i bits appaiono in una sequenza binaria. essa è una rappresentazione del tipo **serial digit-serial bit**.

Se i digit arrivano sequenzialmente mentre i bit sono in parallelo (fig. 3.5.1(b)), si ha una rappresentazione **serial digit-parallel bits**

In fig. 3.5.1(c) i digits sono in parallelo e i bits in serie: si ha quindi una rappresentazione **parallel digit-serizal bits**

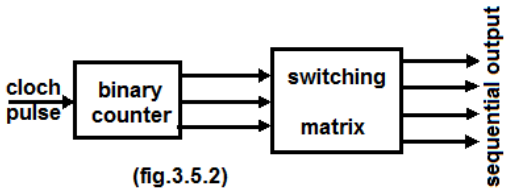
In fig. 3.5.1(d) sia i bits che i digits sono in parallelo. avremo così una rappresentazione **parallel digit.parallel bits**.

In un calcolatore digitale lascelta della rappresentazione dipende da diverse considerazioni quali la velocità di calcolo, il numero dei componenti ect.

Una matrice di commutazione può essere considerata come un filtro di sequernza o un generatore di sequenza.

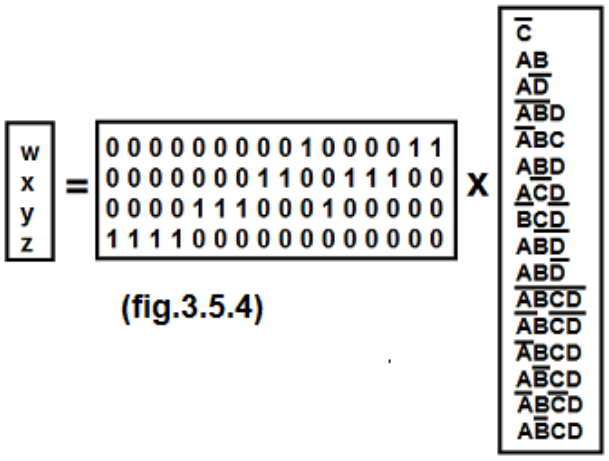
Se la matrice filtra secondo un certo criterio una sequenza di input, viene detta **binary sequence filter**: tali sono ad esempio il full adder ed il traduttore. Per ogni insieme di input viene prodotto come output un altro insieme di sequenze binarie.

Se viene considerato come input di una matrice solo una certa scelta di sequenze binarie e l'uscita è uconosciuta sequenza, si dice che la matrice è un **binary sequence generator**.

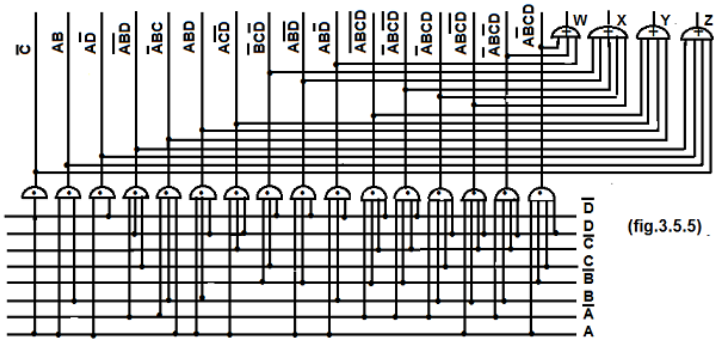


(fig.3.5.2)

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	0	0	1
0	0	1	0	0	1	0	0
0	0	1	1	0	0	0	1
0	1	0	0	0	1	0	1
0	1	0	1	1	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	1	1	0
1	0	0	0	0	1	0	1
1	0	0	1	0	0	1	1
1	0	1	0	0	1	0	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1
1	1	1	0	1	0	0	1
1	1	1	1	0	0	1	1



(fig.3.5.4)



(fig.3.5.5)

In fig. 3.5.2 viene mostrato un generatore di sequenze binarie: un contatore binario conteggia gli impulsi di un orologio: per ogni stato del contatore, la matrice produce un output costituito da un digit BCD

Quindi mentre un contatore binario conteggia, viene generato un numero decimale del tipo **serial-digit parallel bit**.

La matrice descritta in tav.3.5.3 permette di generare la costante π: l'output del contatore binario è rappresentato mediante le lettere A, B, C, D e l'output della matrice dalle lettere W, X, Y, Z.

Entrambi gli output sono espressi in numeri binari. il contenuto delle colonne W, X, Y, Z rappresenta il numero decimale 3141592653589793, dove si assume che la virgola sia posta tra i primi due bit più significativi.

Le quattro funzioni di output possono essere scritte partendo dalla tavola 3.5.3. Se queste funzioni vengono minimizzate, si ottiene la matrice booleana di tav.3.5.4.

La matrice booleana è in forma elementare: in fig. 3.5.5 compare un generatore della costante π sotto forma di **AND-OR MATRIX**.

11.6 Semplificazione delle matrici booleane

Nei paragrafi precedenti è stato mostrato come un circuito di commutazione con output multiplo può essere rappresentato mediante una matrice booleana d forma canonica, di forma elementare o non.

Nel seguito verranno riassunte le caratteristiche di una matrice booleana:

1. Ogni riga della matrice rappresenta una funzione booleana ovvero un output della matrice di commutazione. Il numero delle righe nella matrice non può essere ridotto a meno che una riga non sia composta da soli zeri oppure due o più righe siano identiche.
2. Ogni colonna della matrice rappresenta un prodotto (o una somma). Il numero delle colonne è riducibile e si deve cercare di renderlo minimo.
3. Se una riga della matrice consiste di tutti 0, la riga ed il suo output possono essere tolti dalla matrice.
4. Se una colonna nella matrice consiste di tutti 0, il termine corrispondente nella matrice [n] o [M] può essere eleminato.
5. Se vi è un solo 1 in ogni riga, la matrice di commutazione consiste di sole porte di AND (oppure di OR).
6. Se vi è più di un 1 in almeno una riga, la matrice di commutazione è una matrice **AND-OR** quando la matrice della funzione è della forma [m] oppure una matrice **OR-AND** se la matrice della funzione è della forma [M].
7. Quando un output di una matrice booleana, risulta essere una variabile di un prodotto (o di una somma) di un'altra rappresentazione matriciale, la risultante matrice di commutazione ha due o più livelli.

Con la semplificazione di una matrice booleana, si ottiene di conseguenza quella della matrice di commutazione risultante.

Il procedimento che si segue è quello di minimizzare le singole funzioni booleane con i soliti metodi.

Se la matrice data è in forma canonica e non può essere minimizzata e il numero di bit 1 è superiore a quelli 0, la matrice B booleana complementare risulta più semplice.

La semplificazione può essere ottenuta talvolta esprimendo la matrice in forma non elementare.

11.7 Generazione di segnali di controllo

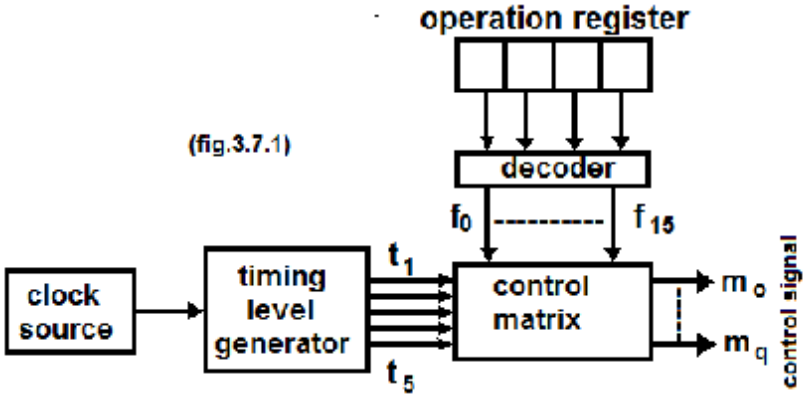
Un calcolatore digitale opera per passi successivi e durante ogni passo viene eseguita una micro-istruzione. Esempi di micro-istruzioni sono gli **schift**, il **conteggio**, il

trasferimento, la **somma**, l **azzeramento**, la **complementazione**. Una sequenza di **micro-istruzioni** costruita per raggiungere un determinato scopo costituisce una **operazione**.

Un calcolatore digitale è progettato per eseguire un certo insieme di operazioni. Queste operazioni son codificate mediante un certo numero di bits: specificando il codice operativo tramite i bits che fanno parte di una voce si può informare il calcolatore di eseguire la corrispondente operazione.Per le istruzioni semplici, ogni istruzione consiste in un codice operativo e in un indirizzo (vedere cap. IV parte II).

Il programmatore scrive una sequenza di istruzioni e questa costituisce un programma capace di risolvere un certo problema.

Quando un calcolatore riceve un'istruzione, il codice operativo contenuto in quest'ultima viene trasferito e memorizzato in un registro di operazioni. Sono necessari allora dei circuiti logici capaci di generare in corrispondenza al codice operativo dei segnali di controllo appropriati che comandano una sequenza di micro-operazioni.



La fig. 3.7.1 mostra una semplice configurazione per la generazione di segnali di controllo.

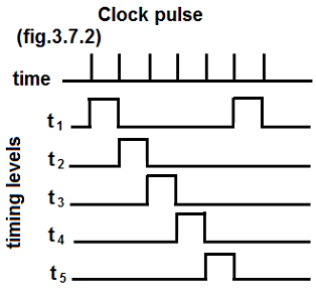
Essa è costituita da un registro, un decodificatore, un orologio, un distributore di livelli temporali, una matrice di controllo.

Supponiamo che in un calcolatore siano previste 16 operazioni e che queste siano codificate con un codice operativo a 4 bit. Siano f_i con I da 0... a 15.

Assumiamo che vi siano 10 micro-operazioni (con cui vengono formate 16 sequenze di micro operazioni rappresentative delle sedici operazioni) e che i segnali di controllo per queste micro-operazioni siano designate con m_j, $j = 0, ..., 9$.

Il fine perseguito dal circuito di fig. 3.7.1 è quello di generare, per ogni codice operativo contenuto nel registro delle operazioni, una corrispondente sequenza di segnali di controllo di micro-operazioni.

Quando la parte del codice operativo di una istruzione viene memorizzata nell'Operation register, il decodificatore a 16 linee di uscita (ognuna per uno dei 16 codici operativi) attiva una e una sola linea.



Nello stesso istante, dei livelli temporali vengono generati mediante un orologio da un **timing level generator**.

Supponiamo che vi siano 5 linee di uscita del generatore di livelli ed indichiamole con t_k $k = 1, ..., 5$: i livelli generati da tali linee vengono mostrati in fig. 3.7.2.

Questi livelli costituiscono una sequenza temporale chiamata **ciclo temporale di base**

Il numero di livelli in un ciclo base è scelto generalmente in modo da poter eseguire una istruzione (quindi controllare una sequenza di micro-operazioni).

Per certe istruzioni quali la moltiplicazione e la divisione spesso vengono usati più di un ciclo base e quindi viene aggiunto un contatore (che non figura in fig. 3.7.1) in modo da conteggiare il richiesto numero di cicli base. L'nput della matrice di controllo è costituito dalle 16 linee f_i, output del codificatore e dalle 5 linee t_k, outputs del generatore di livelli temporali.

La matrice ha 10 linee di uscita e il segnale di ciascuna di queste linee controlla una specifica micro-operazione.

Questi segnali sono rappresentati dalla lettera m_j

Ogni micro-operazione è realizzata dalla combinazione del segnale di controllo e da un segnale di orologio.

Le funzioni relative alla matrice di controllo sono, per esempio, quelle del seguente sistema di equazioni logiche:

$$\begin{aligned} m_0 &= (f_3 t_1 + f_7 t_3 + f_9 t_4)p \\ m_1 &= (f_8 t_1 + f_4 t_2 + f_{12} t_5)p \end{aligned}$$

$$m_9 = (f_{11} t_2 + f_7 t_4 + f_{15} t_5)p$$

dove p rappresenta il segnale dell'orologio. La prima equazione dice che la micro-operazione m_0 è controllata dal codice operativo al livello temporale t_1, o dal codice operativo f_7 al livello t_3, o dal codice operativo f_9 al livello t_4: la micro-operazione viene eseguita quando compare il segnale p dell'orologio. La matrice di controllo del sistema di equazioni logiche è essenzialmente una matrice di commutazione **AND-OR**.

12

Progetto logico di un calcolatore digitale

Nei capitoli precedenti sono stati descritti i circuiti logici capaci di manipolare le informazioni e di memorizzarle, ed è stato presentato un metodo algebrico utile alla rappresentazione di questi circuiti.

Nel presente capitolo si cercherà di presentare il progetto logico di un semplice calcolatore digitale.

12.1 Progetto del calcolatore

Un calcolatore digitale opera mediante delle informazioni, numeriche o di altro tipo, rappresentate in forma digitale.

Dal punto di vista del progetto logico, un calcolatore digitale può essere idealmente descritto come l'insieme de dispositivi di memoria bistabili (per es. flip-flop) connessi da reti logiche.

Lo stato di queste memorie di tipo discreto cambia in certi istanti; lo stato iniziale viene fissato dal programma.

La programmazione di un problema per un calcolatore digitale consiste nella preparazione di una sequenza di istruzioni mediante la quale il problema dato viene risolto dal calcolatore.

Per questo fine, la programmazione comprende l'analisi del problema, la preparazione di un diagramma a blocchi, e la codifica delle istruzioni. Il progetto di un calcolatore digitale può essere diviso in tre fasi: progetto del sistema, progetto logico, progetto dei circuiti.

Nella prima fase vengono prese in considerazione le richieste del problema e le specifiche del calcolatore tenendo conto del costo, delle dimensioni, della manutenzione, ecc. Questa fase richiede siano precisate le operazioni richieste (aritmetiche, logiche, e altre), la velocità delle operazioni, i metodi per il controllo degli errori ed altri dispositivi quali:registri indice, interruttori, ecc., capaci di fornire un uso più efficiente della macchina.

Si può includere in questa fase la selezione del tipo di circuiti logici, il tipo di memoria e le sue capacità, la qualità e la quantità dei dispositivi di **input-output**.

La seconda fase stabilisce il formato della parola, la configurazione della macchina e l istruzioni di macchina; questa fase termina una volta stabilito un insieme di equazioni logiche o un insieme di diagrammi logici dettagliati.

La terza fase comporta il progetto dei circuiti logici, della memoria e dei dispositivi di **ingresso-uscita**.

Il progetto logico di un calcolatore può essere, a sua volta, diviso in tre fasi: progetto funzionale, progetto simbolico, progetto dettagliato. Nella prima fase viene stabilito, innanzi tutto, il formato della parola. Vengono scelti i registri, i contatori, le matrici, gli addizionatori e gli altri elementi; questi vengono completati con la memoria scelta ed i dispositivi di input-output in modo da stabilire un particolare insieme di istruzioni di macchina:

Nella seconda fase viene stabilita la sequenza di controllo del programma e le istruzioni di macchina vengono espresse in istruzioni simboliche. Durante questa fase è utile uno studio dettagliato della sequenza di operazioni relative alle istruzioni di macchina

Nella terza fase viene elaborato un insieme di equazioni logiche od un insieme di diagrammi logici che descrivono le singole le singole operazioni del calcolatore.

Nell'attuale progetto logico di un semplice calcolatore digitale, vengono costruite in modo completo le equazioni logiche; i diagrammi logici verranno mostrati abbastanza dettagliatamente in modo che il diagramma completo possa essere ricavato dal lettore.

12.2 Progetto del sistema

Il calcolatore che verrà illustrato nel seguito fornisce un esempio di progetto logico di un calcolatore digitale capace di memorizzare un programma (stored-program computer) e capace di eseguire un certo numero di operazioni elementari quali la somma, la sottrazione, lo shift a destra e a sinistra di un bit, il trasferimento condizionato e incondizionato. Per semplicità il funzionamento del calcolatore si considera sincrono, binario e parallelo.

12.3 Progetto funzionale

La prima fase del progetto del progetto logico è il progetto funzionale: innanzi tutto viene stabilito il formato della parola.

La configurazione del calcolatore viene formulata scegliendo i registri, le matrici, la memoria ed i dispositivi di **input-output**. Vengono stabilite le istruzioni di macchina e si dà una breve descrizione della sequenza operativa.

12.3.1 Formatto della parola

Una parola (o voce) in un calcolatore è un insieme ordinato di digits che viene trattato dal calcolatore come una unità unitaria.

Si dice che il calcolatore ha una struttura a parole (a voci) se un da e che questa sia piccola. Un numero o una istruzione viene rappresentata mediante una parola.

Per facilitare lo studio del nostro ipotetico calcolatore, supponiamo di fissare la lunghezza della parola e che questa sia piccola. Come vedremo una parola corta consente un numero piccolo di istruzioni di macchina e di indirizzi di memoria.

La lunghezza di parla scelta per questo esempio è di 9 bits:

$$x = x_o x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8$$

Una parola viene trattata dalla unità aritmetica come un numero: in questo caso il formato della parola viene chiamato **formato del numero**.

Sempre in questo esempio, il numero viene dato in una rappresentazione binaria frazionaria con i numeri negativi dati dal complemento a 2 del corrispondente numero positivo.

Il punto binario viene rappresentato idealmente tra i bits x_0 e x_1, mentre x_0 rappresenta il segno.

Poiché il valore di un numero binario con segno è compreso nell'intervallo $-1 \leq x < 1$ il nostro calcolatore viene chiamato **binary fractional computer**.

Quanto detto non costituisce una vera restrizione per i calcoli, poiché un numero fuori dell'intervallo sopra indicato può essere rappresentato nel calcolatore moltiplicandolo per 2^n, con un appropriato valore di n.

La moltiplicazione per 2^n non è altro che una operazione di **shift**.

Poiché il punto binario è fisso, il calcolatore è un **fixed-point computer**.

Esso differisce da un **floating-point computer**, capace di lavorare con una aritmetica in virgola mobile.

operation code part	adress part
$X_0X_1X_2$	$X_3X_4X_5X_6X_7X_8$

La parola considerata come istruzione è suddivisa in due parti (campi): il codice operativo e l'indirizzo. Il formato istruzione del nostro calcolatore sarà quindi:

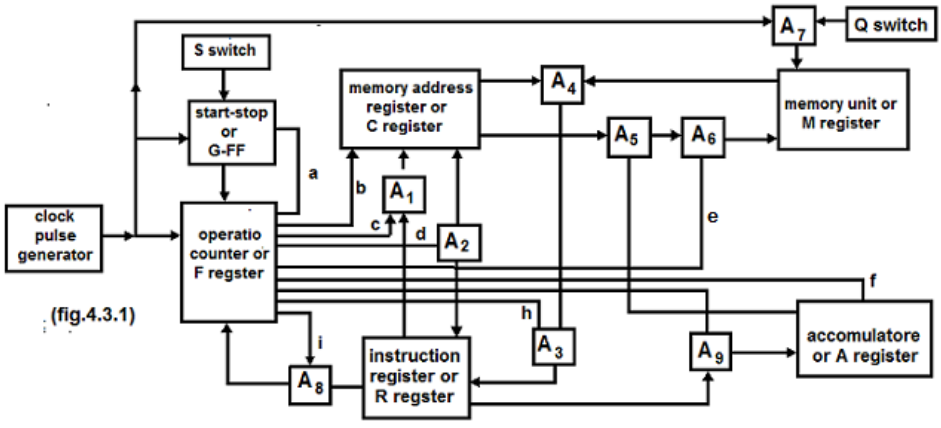
Con un solo indirizzo, questo formato viene chiamato **simple-address format**

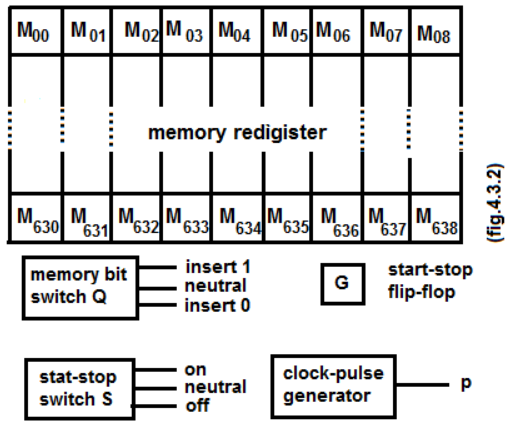
La porzione di tre bit chiamata codice operativo costituisce un codice binario di una operazione di macchina: per es. esso indica un trasferimento, una operazione aritmetica o logica.

I sei bits della parte indirizzo costituiscono il numero dell'indirizzo di memoria di una voce; questa parola di memoria viene chiamata operando. Un tale formato di istruzione indica che una operazione (specificata dalla porzione di codice operativo) viene eseguita sull'operando conservato nella posizione di memoria specificata dalla porzione di indirizzo.

Certe istruzioni non necessitano di operandi:in questo caso la parte relativa all'indirizzo può essere usata per altri scopi.

12.3.2 Configurazione del calcolatore





La configurazione del nostro calcolatore ipotetico è mostrata in fig. 4.3.1.

Per essere uno **stored-programm computer** il calcolatore deve memorizzare sia i numeri che le istruzioni.

Questa considerazione fa introdurre l'unità di memoria (indicata con i registri M): la sua capacità è di 64 parole di 9 bits ciascuna.

Come mostrato in figura 4.3.2, questi flip-flop sono denotati con $M_{(ji)}$, dove j indica la J$^{\text{sima}}$ parola e I il bit$^{\text{simo}}$.

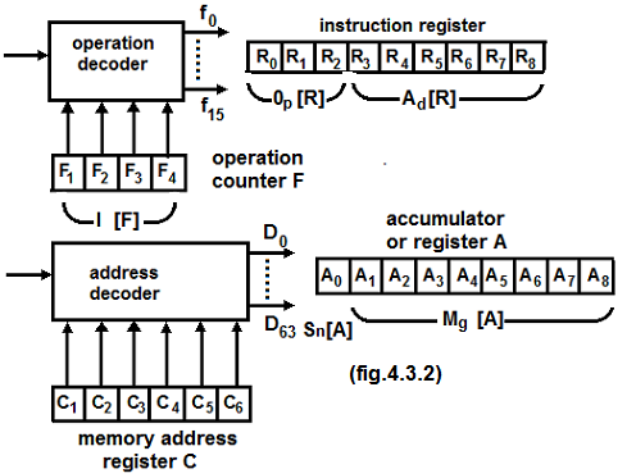
Ad ogni locazione di memoria viene attribuito un indirizzo: la selezione degli indirizzi viene fatta mediante il **memory register C**.

Il registro C consiste di 6 flip-flop (fig. 4.3.2).

Un duplicatore di indirizzi viene messo tra il registro C e la memoria.

Perché il calcolatore possa eseguire operazioni aritmetiche, occorre introdurre una unità aritmetica, capace in questo caso semplice di sole somme e sottrazioni.

Un accumulatore (registro A) è sufficiente come unità aritmetica: esso consiste in 9 bits, in modo da memorizzare una intera parola (fig. 4.3.2)



Quando una istruzione viene presa dalla memoria, essa viene conservata temporaneamente in un registro per essere successivamente eseguita. L'apposito registro viene chiamato **intruction register** (R) ed è ovviamente di 9 bits

I bit $R_0R_1R_2$ memorizzano il codice operativo, mentre i 6 bits da R_3 a R_9 conservano la parte indirizzo della parola istruzione.

La necessità di eseguire un controllo di sequenze introduce la **program control unit**.

Il nostro calcolatore ha un **operation counter** (F), di 4 bits (fig. 4.3.2), I tre bits f_2 f_3 f_4, danno il codice operativo.

L'operation counter conteggia gli impulsi di un orologio e genera i segnali di controllo mediante un decodificatore di operazioni.

Un generatore di impulsi d'orologio da una sequenza di impulsi temporali con periodo τ.

L'impulso di orologio serve a sincronizzare le operazioni del calcolatore.

Un altro clock-pulse generator (non indicato in fig. 4.3.1) genera un singolo impulso: esso è sotto il controllo dell'operatore che in questa maniera, può osservare il procedere del calcolatore passo per passo.

Altro dispositivo di controllo è il flip-flop G, capace di avviare od arrestare le operazioni del calcolatore.

Il controllo del flip-flop G è accessibile all'esterno mediante lo switch (interruttore) S.

Come dispositivi di input capaci di inserire in memoria dati e istruzioni, vi sono gli switch Q mentre per fornire i risultati ci possono essere dei dispositivi luminosi (non indicati in fig. 4.3.1). Sia gli switchs Q che i dispositivi luminosi sono connessi ai singoli bit di memoria.

I dispositivi luminosi sono anche connessi ai singoli bit dei registri in modo da osservare le operazioni del calcolatore durante eventuali test.

12.3.3 Istruzioni di macchina

Formuliamo ora l'insieme di istruzioni che la macchina è capace di eseguire.

La lista di istruzioni capace di risolvere un certo problema deve essere ovviamente codificata prima di essere caricata nel calcolatore in modo che le istruzioni stesse siano comprensibili alla macchina:

Le istruzioni eseguibili da questo calcolatore ipotetico sono raggruppate in tabella 4.3.3 seguente

Codice	Istruzione	Descrizione dell'istruzione
000 uvwxyz	Addizione	Prende il numero dalla memoria all'indirizzo uvwxyz, lo mette nel registro R; somma il contenuto di R al contenuto del registro A e conserva il risultato nel registro A.
001 uvwxyz	Sottrazione	Prende il numero dalla memoria all'indirizzo uvwxyz, lo mette nel registro R; sottrae il contenuto di R al contenuto del registro A e conserva il risultato nel registro A.
010 uvwxyz	trasferimento condizionato	Se $A_0 = 1$ (segno di A negativo) si prende come nuova istruzione quella contenuta nella memoria all'indirizzo uvwxyz; se $A_0 = 0$ (segno di A positivo), si prende la nuova istruzione in sequenza.
100 uvwxyz	trasferimento incondiziona-to	Prende la nuova istruzione dalla memoria di indirizzo uvwxyz.
011 uvwxyz	Memorizzazione	Memorizza il contenuto del registro A nell'indirizzo uvwxyz di memoria
1011000 yz	Shift a destra	Il contenuto del registro A viene shiftato a destra di 1 bit.-
1010100 yz	Shift a sini-stra	Ilo contenuto del registro A viene shiftato a sinistra di 1 bit.
1010010 yz	Azzera l'accu-mulatore	Si azzera l'accumulatore A
1010001 yz	Stop	Stop della macchina e nel registro C compare l'indirizzo 000000.

Poiché la macchina non compie alcuna operazione in relazione agli organi di input-output, non sono previste istruzioni relative ad operazioni di ingresso-uscita.

La quantità uvwxyz indica uno dei 64 possibili indirizzi di memoria.

È da osservare che le ultime quattro istruzioni di tabella 4.3.3 non fanno riferimento a posizioni di memoria (cioè non è necessario un operando e quindi la parte indirizzo può essere utilizzata nella codificazione del codice operativo).

Quindi, pur avendo solo 3-bit a disposizione del codice, si hanno in effetti nove operazioni.

12.3.4 Sequenza operativa del calcolatore

Facendo riferimento alla configurazione di fig-4.3.1, le istruzioni codificate che formano un programma vengono inserite per prima cosa in memoria tramite l'interruttore Q attraverso la porta A_7.

L'operatore pone l'interruttore S in posizione **ON** e la macchina incomincia a funzionare.

Lo stato di tutti i registri viene cambiato tramite i segnali di controllo provenienti dallo **operation counter**.

Questi cambiamenti possono solo aver luogo in concomitanza degli impulsi di orologio.

Quando la macchina opera per la prima volta, il contenuto del registro C è 000000; per cui la prima parola di memoria è la prima istruzione.

Il segnale di controllo h trasferisce questa istruzione dall'unità di memoria al registro R tramite le porte A_3 e A_4.

La parte codice operativo della istruzione che si trova ora nel registro R viene trasferita al registro F tramite la porta A_8 sotto il controllo del segnale I, mentre il segnale di controllo C trasferisce la parte indirizzo tramite la porta A_1 al registro C.

Il codice operativo nel registro F è ora decodificato e vengono generati i comandi necessari all'esecuzione dell'istruzione,

Per le operazioni aritmetiche, il segnale di controllo d trasferisce, attraverso le porte A_3 e A_4, un numero dalla memoria al registro R, ed il segnale di controllo G determina, mediante la porta A_9, la realizzazione di una specifica operazione aritmetica sul numero contenuto in R e su quello inizialmente in A.

Per le operazioni di **shift** e di azzeramento dell'accumulatore si ha il segnale di controllo f.

L'operazione di trasferimento relativa alla sequenza di controllo avviene sotto i segnali B o H, mentre il segnale E, selezionando la porta A_6, conserva mediante la porta A_5, il controllo dell'accumulatore nella memoria all'indirizzo che si trova nel registro C.

L'ultima istruzione del programma in memoria è in genere una istruzione di **stop**: l'**operation counter** invia il segnale di controllo A al flip-flop G e la macchina si ferma.

12.4 Fase di progettazione simbolica

In questa fase viene stabilita la sequenza di controllo del programma e le operazioni di macchina vengono rappresentate mediante espressioni simboliche.

Tra i simboli già introdotti nel Cap. 3 Parte II, dobbiamo introdurre il seguente: $< >$, con cui indichiamo l'indirizzo di una parola di memoria e precisamente $M < C >$ indica la locazione di memoria il cui indirizzo è nel registro C.

Le istruzioni indicate in tabella 4.3.3 vengono ora espresse in termini di espressioni simboliche elencate nella seguente tabella 4.4.1

Istruzioni	Operazioni richieste	Operazioni ausiliarie
Addizione (000uvwxyz)	$(M < C >=> R; (R) + (A) => A$	$(Ad[R]) => C; (C) + 10 => C$
Sottrazione (001uvwxyz)	$(M < C > R => R;$ $(A) - (R) => A$	come sopra
Trasferimento condizionato (010uvwxyz)	nessuno	come sopra se $A_0 = 0$ nessuno se $A_0 = 1$
Memorizzazione (011uvwxyz)	$(A) => M < C >$	$(Ad[R]) => C; (C) + 1 => C$
Trasferimento incondizionato (100uvwxyz)	nessuno	nessuno
Shift right (1011000)	$(A_{i-1} => A_i; (A)_0 => A_0$	$(Ad[R])? > C; (C) + 1 => C$
Test Shift left (1010100yz)	$(A_i + 1) => A_i; (A_0) => A_8$	come sopra
Azzeramento accumulatore (1010010yz)	0=>A	come sopra
Stop (10010001yz)	$0 => G;$ $0 => C;$ $1 => F$	nessuna
Comando d'avvio (manuale)	1=>G	nessuna
Comando inserimento dati (manuale)	$(Q_{ij}) => M_{ij}$	nessuna

L'addizione mostrata in tabella 4.3.3 è costituita da due operazioni La prima operazione consiste nel "prendere un numero dalla memoria di indirizzo **uvwxyz** e metterlo nel registro R", simbolicamente.

$$(M < C >) \Rightarrow R$$

dove C contiene l'indirizzo di memoria **uvwxyz**.

La seconda operazione è: "addizionare il contenuto del registro R al contenuto del registro A, e conserva il risultato nel registro A. Simbolicamente si ha:

$$(R) + (A) \Rightarrow A$$

Nella tabella 4.4.1 compaiono le due espressioni simboliche per la istruzione di addizione. Si hanno inoltre due espressioni simboliche simili alle precedenti, per la sottrazione.

L'istruzione di memorizzazione del contenuto del registro (A) nella memoria di indirizzo **uvwxyz**, si esprime come:

$$(A) \Rightarrow M < C >$$

dove C contiene il riferimento all'indirizzo di memoria **uvwxyz**.

L'istruzione di **shift** a destra consiste nello spostare il contenuto del registro A di un bit a destra e simbolicamente:

$$(A_{i-1}) \Rightarrow A_i \qquad dove\ i = 1, ..., 8$$

$$(A_0) \Rightarrow A_0$$

La seconda espressione dice che lo **shift** a destra richiede che il bit più a sinistra del registro A conservi il valore originale durante l'operazione di scorrimento.

L'operazione di **shift** a sinistra richiede che il contenuto del registro A sia spostato di 1 bit a sinistra:

$$(A_{i+1}) \Rightarrow A_i \qquad \text{dove } i = 1, ..., 7$$
$$(A_0) \Rightarrow A_8$$

Questa operazione di **shift** a sinistra muove il bit più a sinistra del registro A in maniera circolare.

L'azzeramento dell'accumulatore diventa:

$$0 \Rightarrow A$$

L'istruzione di **stop** consiste nell'arrestare la macchina e porre nel registro C il primo indirizzo 000000; simbolicamente:

$$0 \Rightarrow G \qquad e \qquad 0 \Rightarrow C$$

L'operazione di avvio del calcolo e di inserimento delle istruzioni dei dati in memoria non vengono indicate in tabella 4.4.1 in quanto operazioni manuali.

Simbolicamente, esse sono:

$$1 \Rightarrow G \qquad e \qquad (Q_{ji}) \Rightarrow M_{ji}$$

La prima espressione indica il set del flip-flop G mentre la seconda indica l'inserimento del contenuto dell'interruttore Q_{ji} nel bit di memoria M_{ji}

12.4.1 Ciclo d'istruzione e ciclo di esecuzione

Sia le istruzioni che i dati di un programma capace di risolvere un certo problema vengono inizialmente conservati in memoria.

Il programmatore scrive queste istruzioni in una sequenza in accordo con l'ordine degli indirizzi di memoria in cui queste istruzioni verranno memorizzate.

Nel nostro calcolatore ipotetico la prima istruzione del programma viene messa nella posizione 000000.

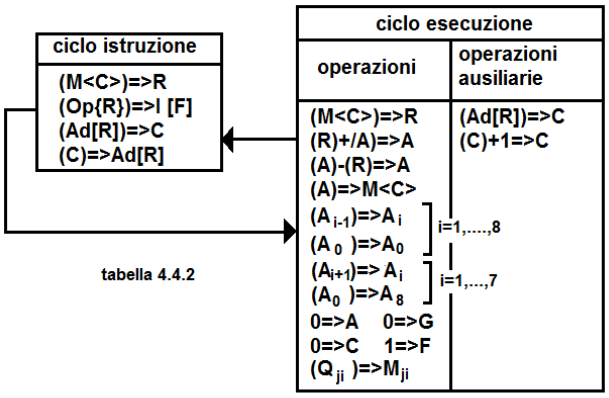
Il calcolatore esegue la sequenza di istruzioni conservata in memoria: esso opera secondo una sequenza interna eseguendo ogni istruzione come vengono chiamate, una dopo l'altra, dal programma.

La sequenza interna passa da un ciclo d'istruzione ad un ciclo di esecuzione.

Durante il **ciclo istruzione** viene presa una istruzione dalla memoria e il calcolatore viene posto nelle condizioni di poter compiere la esecuzione di una sola istruzione.

Durante il **ciclo esecuzione** viene eseguita l'istruzione ed il calcolatore viene posto nelle condizioni di poter tornare al **ciclo istruzione**.

In fig-4.4.2 sono raffigurati entrambi i cicli.



Durante la fase del ciclo di istruzione vi sono quattro operazioni (fig-4.4.3a); la prima permette di prendere una istruzione: una parola viene presa dalla memoria il cui indirizzo è nel registro C (000000 per la prima istruzione) e trasferita nel registro R; simbolicamente:

$$(M < C >) \Rightarrow R$$

Il contenuto di C (memory-address) durante questa operazione è un **instruction address**.

La seconda e terza operazione (che avvengono contemporaneamente) sono operazioni di trasferimento mediante le quali vengono trasferite la porzione del codice operativo dal registro R, Op[R], nella porzione istruzione del registro F, e la parte indirizzo di R, $Ad[R]$ nel registro C:

$$Op[R] \Rightarrow I[F]$$
$$Ad[R] \Rightarrow C$$

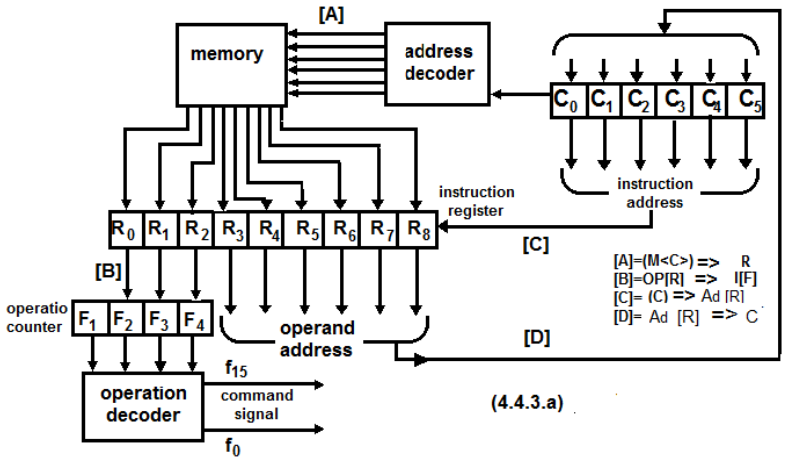
A questo istante la la parte indirizzo di R, è un **operand address** e quindi non è più l'indirizzo di istruzione menzionato nella fase precedente.

Dopo il trasferimento la parte di codice operativo fa generare dal decodificatore di operazioni, dei segnali di comando, dando così inizio all'esecuzione della istruzione mentre l'indirizzo dell'operando si trova ora nel registro R:

Normalmente questo **ciclo istruzione** è completo: nel nostro calcolatore ipotetico, data la sua semplicità, è necessaria una quarta operazione che trasferisca l'indirizzo della istruzione contenuta in C nella parte indirizzo di R:

$$(C) \Rightarrow Ad[R]$$

Il motivo di questa ulteriore operazione è dovuto al fatto che la parte indirizzo del registro R (operand address) viene trasferita nel registro C e quindi l'indirizzo della istruzione contenuta in C (000000 per la prima istruzione) viene perduto.



Questo indirizzo della istruzione che è stato preso in considerazione deve essere conservato in qualche maniera in quanto l'indirizzo della nuova istruzione è ottenuto da quello della presente, sommando 1.

Poiché la parte indirizzo del registro R è stata trasferita nel registro C e non è usata per il momento. l'indirizzo della presente istruzione può essere trasferito per una memorizzazione temporanea nella parte indirizzo del registro R.

In altre parole, l'indirizzo della presente istruzione contenuto in C viene trasferito in R e nello stesso istante l'indirizzo dell'operando contenuto i R viene trasferito in C.

Questi due trasferimenti simultanei sono possibili se si fa l'ipotesi che i flip-flop dei registri R e C abbiano un delay sufficiente.

In un calcolatore digitale ordinario con capacità di memorizzazione di programmi c'è generalmente un ulteriore contatore di istruzioni atto a memorizzare l'indirizzo della attuale istruzione e, tramite incremento di 1 unità dopo l'esecuzione di una istruzione si determina l'indirizzo della nuova istruzione da eseguire.

In tal caso non è necessaria l'operazione di trasferimento da C ad Rprecedentemente descritta.

Le operazioni che avvengono durante il **ciclo istruzioni** sono sempre le stesse, mentre quelle che compaiono durante il **ciclo esecuzione** dipendono dal codice operativo dell'istruzione.

In fig-4.4.3b si è preso in considerazione il ciclo esecutivo di una istruzione d'addizione: esso incomincia prendendo la parola operando dalla memoria localizzata dall'**operand address** contenuto in C. La parola operando è trasferita nel registro R che viene utilizzato attualmente come memoria temporanea:

$$(M < C >) \Rightarrow R$$

Durante l'operazione di caricamento dell'operando, il contenuto della parte indirizzo di R che altro non è se non l'indirizzo della attuale istruzione, viene trasferito in C:

$$Ad[R] \Rightarrow C$$

L'operando ora contenuto in R viene addizionato al contenuto del registro $< a$ (che è un accumulatore):

$$(R) + (A) \Rightarrow A$$

Infine, il contenuto del registro C (contenente l'indirizzo della attuale istruzione) viene incrementato di 1 diventando così l'indirizzo della nuova istruzione:

$$(C) + 1 \Rightarrow C$$

Così il ciclo esecutivo risulta completo.

La nuova istruzione è pronta per essere presa dalla memoria mediante la prima operazione del **ciclo istruzione** dell'indirizzo di memoria indicato dal contenuto di C.

Nel caso di una istruzione di trasferimento incondizionato, l'indirizzo della nuova istruzione è dato dall'indirizzo dell'operando.

Nel caso di una istruzione di trasferimento condizionato, la sequenza procede come nel caso del trasferimento incondizionato, se la condizione è verificata ($A_0 = 1$); se la condizione non è verificata ($A_0 = 0$), la nuova istruzione da eseguire viene determinata mediante l'operatore $(C) + 1 => C$.

L'operazione ausiliaria $(1 => F)$ viene spiegata nel seguito.

12.4.2 Diagramma di stato

L'unità di controllo del programma (registro F) del nostro calcolatore ipotetico consiste in un contatore e in un decodificatore associato (fig-4.3.2).

Gli stati del contatore rappresentano gli stati del calcolatore; quindi la sequenza degli stati del contatore controlla la sequenza operativa del calcolatore.

Poiché sono contemplate nove operazioni, sono richiesti all'**operation counter** nove stati, e ciò implica un minimo di 4 Flip-Flop, f_1, f_2, f_3, f_4.

I flip-flop f_2 e f_4 costituiscono la parte istruzione del registro F(o I[F]); in essi vengono memorizzati i 3 bit della porzione codice operativa del registro istruzioni R.

Per un contatore con quattro flip-flop, si possono presentare 16 stati; essi vengono indicati con la notazione f_i in tabella 12.1 (l'indice I rappresenta il numero binario corrispondente di 4 bit).

Ogni stato rappresenta 1 segnale di comando, come si vede da tavola 12.1.

Poiché sono sufficienti 14 stati, i comandi f_6 e f_7 non vengono usati. Gli stati f_4 e $f_10$sono quelli corrispondenti al ciclo istruzione; gli stati rimanenti corrispondono al ciclo esecutivo.

Poiché vi sono 9 istruzioni, vi sono nove possibili percorsi da f_10 a f_4; durante il ciclo esecutivo, la sequenza operativa segue una di queste nove vie.

Quando il calcolatore esegue un programma, non fa altro che percorrere ciclicamente queste vie del diagramma di stato (fig-4.4.5) in accordo con le istruzioni programmate.

La scelta dei 14 segnali di comando tra i 16 possibili stati del contatore non è del tutto arbitraria ma è stata fatta cercando di semplificare la logica del circuito.

Infatti gli stati f_0, f_1, f_2, f_3, f_4, f_5 sono relativi a comandi di istruzioni e questa scelta fa si che il flip-flop f_1 sia uguale a 0 per tutti i comandi di istruzione, semplificando così la logica del circuito.

Mediante la tabella 4.4.4 e conoscendo la sequenza di controllo relativa ai cicli-istruzione e cicli.esecuzione si può costruire la figura 4.4.5 detta anche diagramma

Tabella 12.1: Stati dello operation counter

operation counter f_1 f_2 f_3 f_4	stato	segnali di comando
0 0 0 0	f_0	comando di addizione
0 0 0 1	f_1	comando di sottrazione
0 0 1 0	f_2	comando di trasferimento condizionale
0 0 1 1	f_3	comando di immagazzinaggio
0 1 0 0	f_4	comando ciclo istruzioni
0 1 0 1	f_5	comando per f_{12}, f_{13}, f_{14}, f_{15}
0 1 1 0	f_6	non usato
0 1 1 1	f_7	non usato
1 0 0 0	f_8	comando operazione di addizione
1 0 0 1	f_9	comando operazione di sottrazione
1 0 1 0	f_{10}	comando ciclo istruzioni
1 0 1 1	f_{11}	comando operazione aggiunta di 1
1 1 0 0	f_{12}	comando di scorrimento a destra
1 1 0 1	f_{13}	comando di scorrimento a sinistra
1 1 1 0	f_{14}	comando di svuotamento dell'accumulatore
1 1 1 1	f_{15}	comando di arresto

di stato in cui compaiono gli stati del calcolatore dopo le singole operazioni e le operazioni conseguenti ad ogni singolo stato.

In figura 4.4.5 l'operazione di prendere una istruzione in memoria durante il ciclo-istruzione avviene allo stato f_4, mentre le altre tre operazioni relative al ciclo-istruzione avvengono allo stato f_10.

La sequenza interna del ciclo esecutivo dipende dal contenuto dei flip-flop R_0, R_1, R_2.

Vi sono sei strade: per l'addizione, la sottrazione, il trasferimento condizionato, il trasferimento incondizionato, il comando di memorizzazione, inoltre vi sono le strade che compaiono quando $R_0R_1R_2$ è 101 e indicano le operazioni di **left-shift**, **right-shift**, **clear accumulator** e **stop**.

L'operazione che costruisce l'indirizzo relativo all'istruzione successiva (detto anche indirizzo di ritorno) avviene negli stati f_0, f_1, f_2, f_3 e f_5 mentre non è necessaria se l'istruzione appena eseguita è un trasferimento incondizionato.

Uno zero viene inserito nel flip-flop G allo stato f_5 se C_3 è 1.

L'operazione di incremento di 1 avviene durante lo stato f_11.

L'operazione di prendere dalla memoria l'operando avviene durante gli stati f_0 e f_1 mentre le operazioni di addizione e di sottrazione vengono eseguite agli stati f_8 e f_9 rispettivamente.

Allo stato f_15, il comando di stop inserisce tutti 0 nel registro C e tutti 1 nel registro F.

L'**operation counter** continua a conteggiare i segnali dell'orologio ma il suo stato rimane f_15 a causa dell'operazione $1 => F$:

Perciò, sotto il comando di stop il calcolatore non si arresta ma viene sospesa solo la sequenza operativa.

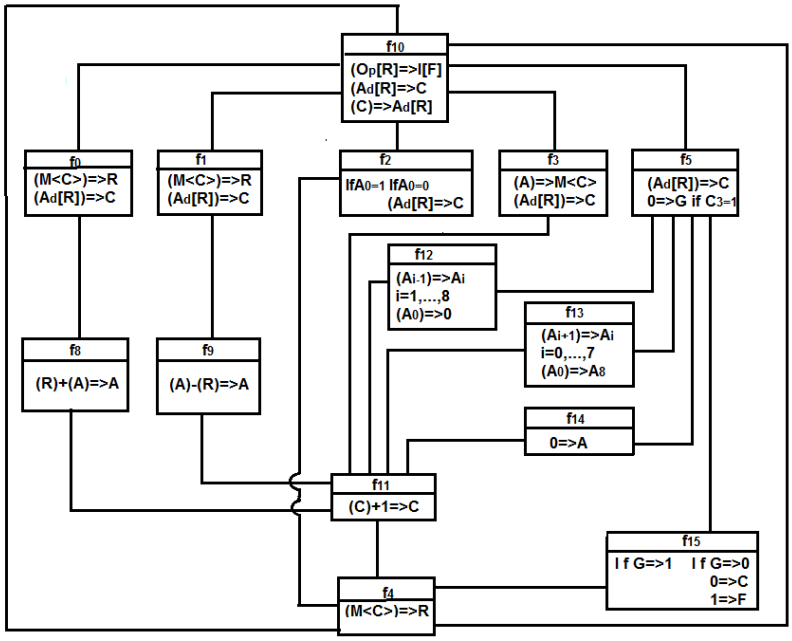
Quando lo switch S viene messo nello stato **off**, il flip-flop G assume lo stato 0: il calcolatore continua ad eseguire l'istruzione in corso fino a quando l'**operation counter**, raggiunge lo stato f_4.

A questo punto, l'operation counter assume lo stato $f_1 5$ e rimane in tale stato.

Quando la sequenza è stata sospesa, può essere ripristinata ponendo lo switch S nella posizione **ON**

Quando questo accade, il Flip-Flop G assume lo stato 1 e l'operation counter assume lo stato f_4

Poiché il contenuto del registro C era stato cambiato in 000000, il calcolatore comincia dall'istruzione contenuta nella prima posizione di memoria.



12.5 Equazioni logiche

Durante questa fase vengono derivate le equazioni logiche dei circuiti, partendo dalle espressioni simboliche di figura 4.4.5.

Queste equazioni rappresentano le equazioni di **input** dei Flip-flop dei registri F, A, C, R, M e del Flip-Flop G.

12.5.1 Operation counter

Nell'operation counter vi sono quattro flip-flop f_1, f_2, f_3, f_4: esso deve realizzare la sequenza di figura 4.4.5 ed assumere lo stato 1111 quando G è 0.

Da figura 4.4.5 si ha allora che lo stato del flip-flop f_1 cambia quando dallo stato f_4 (0100) si passa allo stato $f_1 0$ (1010).

Cambia inoltre quando $f_1 0$ (1010) diventa f_0 (0000), con R_0, R_1, R_2 (000).

In breve lo stato del flip-flop f_1 cambia quando si presenta una delle sei seguenti condizioni.

$$f_1 0 \bar{R}_0 \bar{R}_1 \bar{R}_2 + f_1 0 \bar{R}_0 \bar{R}_1 R_2 + f_1 0 \bar{R}_0 R_1 \bar{R}_2 + f_1 0 \bar{R}_1 R_2 + f_1 0 R_0 \bar{R}_1 R_2 + f_1 0 R_0 \bar{R}_1 \bar{R}_2 = f_1 0$$

Le altre condizioni che determinano il cambiamento dello stato di f_1 possono essere dedotte con considerazioni simili, tenendo sempre presente il diagramma di stato di figura 4.4.5.

Chiamando quindi p l'impulso d'orologio che commuta i flip-flop, si hanno le seguenti equazioni di **input** per f_1, f_2, f_3, f_4.

$$f_{1t} = [f_4 + f_{10} + f_0 + f_1 f_2 \bar{A}_0 + f_3 + f_5(C_0 + C_1 + C_2 + C_3) + f_{11} + f_{15}G]p$$
$$f_{2t} = [f_4 G + f_{10} R_0 + f_2 A_0 + f_{11} + f12 + f_{13} + f_{14}]p$$
$$f_{3t} = [f_4 + f_{10}\bar{R}_1 + f_2 A_0 + f_5(C_2 + C_3) + f_8 + f_9 + f_{11} + f_{12} + f_{13} + f_{15}G]p$$
$$f_{4t} = [f_4 \bar{G} + f_{10} R_2 + f_2 \bar{A}_0 + f_5(C_0 + C_2) + f_8 + f_{11} + f_{12} + f_{14} + f_{15}G]p$$

F_i	$F_i(\tau)$	f_{it}
0	1	1
1	1	0

Quando il contatore si trova nello stato f_15 e il flip-flop G ha valore 0, viene richiesta l'operazione 1 => F: in tavola 4.5.2 vengono mostrati i valori assunti dai flip-flop f_i durante tale operazione.

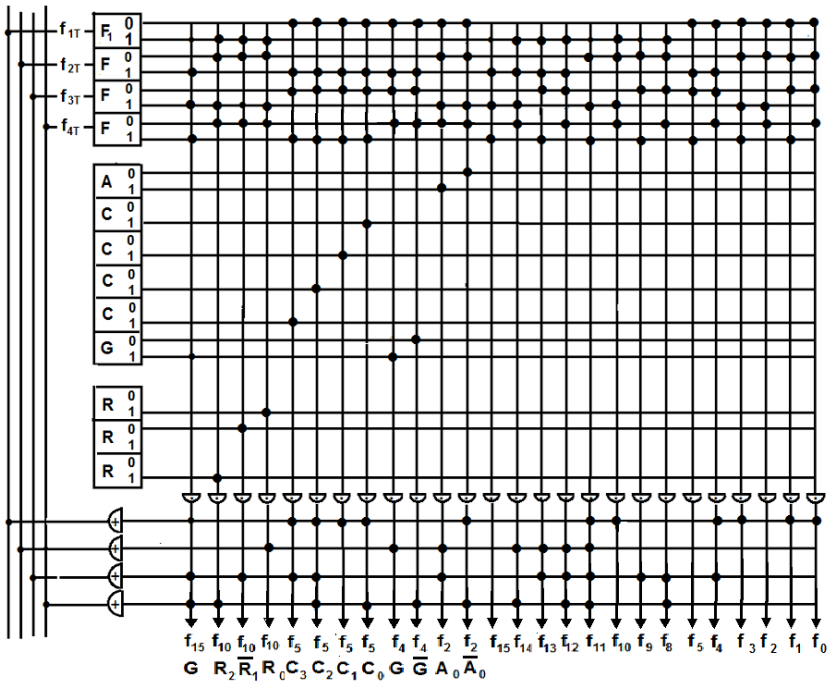
Le equazioni di input dei flip-flop F in tale caso saranno:

$$f_{it} = \bar{F}_i G f_{15} p \qquad i = 1, ...4 \quad (4.5.3)$$

Poiché f_15 è $f_1\ f_2\ f_3\ f_4 = 1111$, neg f_i f_15 nella equazione 4.5.3 è sempre 0, e quindi l'equazione (4.5.3) non è necessaria.

Le equazioni 4.5.1 sono le equazioni di input del registro F-

In fig. 4.5.4 compare il diagramma logico relativo alle (4.5.1).



12.5.2 Accumulatore

Come mostrato nel diagramma 4.4.5, l'accumulatore interviene durante l'esecuzione delle operazioni relative agli stati:

$$f_8 : [R] + (A) \Rightarrow A$$
$$f_9 : (A) - (R) \Rightarrow A$$
$$f_{12} : (A_0) \Rightarrow A_0 \quad (A_{i-1}) \Rightarrow A_i \quad i = 1, ..., 8$$
$$f_{13} : (A_0) \Rightarrow A_8 \quad (A_{i+1}) \Rightarrow A_i \quad i = 0, ..., 7$$
$$f_{14} :=\Rightarrow A$$

Durante l'operazione di addizione relativa allo stato f_8, A_i e R_i sono gli i$^{\text{simi}}$ bits degli addendi.

K_i è il riporto dovuto alla somma fatta sugli (i+1)$^{\text{simi}}$ bits.

L'operazione di somma viene mostrata in tabella 4.5.5.

A_i	R_i	K_i	K_{i-1}	A_{it}	Q_{it}
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	1	0

Da tale tabella, si ricavano le seguenti equazioni di input:

$$Q_{it} = (R \oplus K_i) f_8 p \quad i = 0, ..., 8 \qquad (4.5.6)$$

mentre per il riporto si ha:

$$K_{i-1} = (R_i K_i + R_i A_i + A_i K_i) f_8 P \quad i = 1, ..., 8$$
$$K_8 = 0$$

Per l'operazione di sottrazione relativa allo stato f_9, A_i e R_i sono rispettivamente l'i$^{\text{simo}}$ bit del minuendo e dell'i$^{\text{simo}}$ bit del sottraendo.

Le equazioni di input del riporto sono le stesse di (4.5.6) ad eccezione di R_i che viene sostituito da $\bar{R}_i$ e K_8 viene sostituito da 1in modo da sostituire il contenuto di R con il suo complemento a 2.

Le equazioni di input e del riporto sono quindi:

$$a_{it} = (\bar{R}_i \oplus K_i) f_9 p \quad i = 0, ..., 8 \quad (4.5.7)$$
$$K_{i-1} = \bar{R}_i K_i + \bar{R}_i A_i + A_i K_i) f_9 p \quad i = 1, .., 8$$
$$K_8 = f_9 p$$

L'operazione di scorrimento a destra avviene allo stato $f_1 2$.

Esso fa scorrere il contenuto dell'accumulatore di un bit a destra mentre lo stato di A_0 rimane inalterato.

La tabella (4.5.8) rappresenta l'operazione di scorrimento a destra.

Le equazioni di input sono:

A_{i-1}	A_i	$A_i(\tau)$	A_{it}
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

$$a_{it} = (A_{i-1} \oplus A_i f_{12} p \quad i = 1, ..., 8$$

Non vi è alcuna equazione pera_{0t} in quanto lo stato del flip-flop A_0 non deve cambiare.

L'operazione di scorrimento a sinistra avviene allo stato $f_1 3$.

Essa fa scorrere il contenuto dell'accumulatore di un bit verso sinistra mentre lo stato di A_8 viene sostituito da quello di A_0.

A_i	$A_1(\tau)$	A_{it}
0	0	0
1	0	1

Le equazioni di input saranno:

$$a_{it} = (A_i \oplus A_{i+1} f_{13} p \quad i = 0, ..., 7$$
$$a_{8t} = (A_0 \oplus A_8 f_{13} p$$

L'operazione di azzeramento dell'accumulatore avviene allo stato $f_1 4$ (tabella 4.5.9):

Le equazioni di input sono:

$$a_{it} = A_i f_{14} p \quad i = 0, ..., 8$$

quindi le equazioni di input dei flip-flop che costituiscono l'accumulatore a saranno:

$$a_{it} = (R_i p \oplus K_i) f_8 p + (\bar{R}_i \oplus K_i) f_9 p + (A_{i-1} \oplus A_i) f_{12} p + (A_i \oplus A_{i+1} f_{13} p + A_i f_{14} p$$
$$i = 1, ..., 7$$
$$a_{0t} = (A_0 \oplus K_0) f_8 p + (\bar{R}_0 \oplus K_0) f_9 p + (A_0 \oplus A_1) f_{13} p + a_0 f_{14} p$$
$$a_{8t} = (A_8 \oplus K_8) f_8 p + (\bar{R}_8 \oplus K_8) f_9 p + (A_7 \oplus A_8) f_{12} p + (A_0 \oplus A_8) f_{13} p + a_8 f_{14} p$$
$$K_{i-1} = (R_i K_i + R_i A_i + A_i K_i) f_8 p + (\bar{R}_i K_i + |bar R_i A_i + A_i K_i) f_9 p$$
$$i = 1, ..., 8$$
$$K_8 = f_9 p$$

Il diagramma logico dell'i_{simo} stadio dell'accumulatore viene presentato in fig. 4.5.10

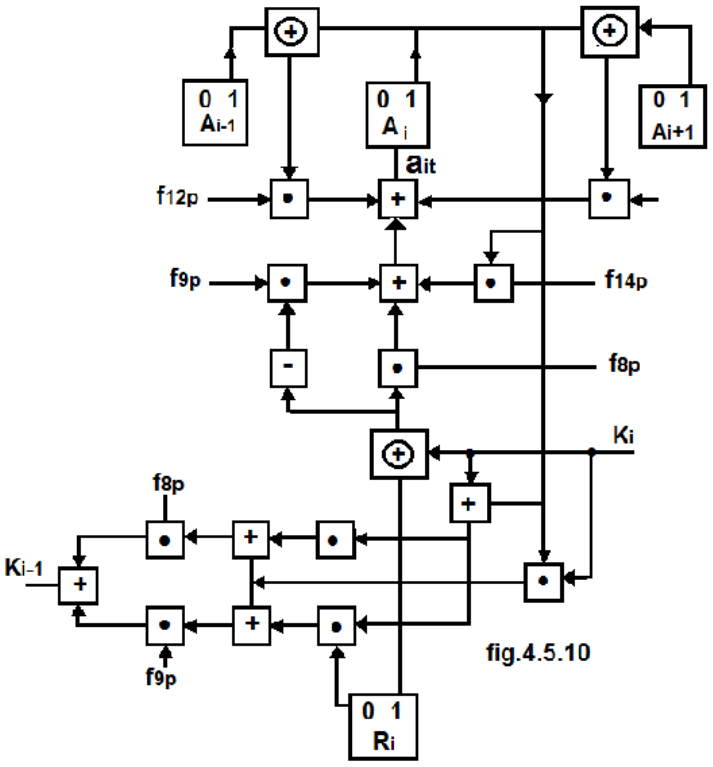


fig.4.5.10

12.5.3 Memory address register

Nel diagramma di stato di fig-4.4.5, il registro C deve permettere le seguenti operazioni:

$$f_{10} + f_0 + f_1 + f_2\bar{A}_0 + f_3 + f_5 \quad : (Ad[R]) \Rightarrow C \quad : (C) + 1 \Rightarrow C \quad : 0 \Rightarrow C$$

La tabella della verità relativa alla operazione $(Ad[R] => C$ è quella posta a fianco.

Le equazioni di input da tale tavola sono:

$$C_{it} = (R_{i+3} \oplus C_i)gp \quad i = 0, ..., 5$$
$$g = f_{10} + f_0 + f_1 + f_2\bar{A}_0 + f_3 + f_5$$

L'operazione $(C) + 1 => C$ avviene allo stato $f_1 1$: durante questa operazione, il registro C funziona come un contatore binario.

Le equazioni di input saranno quindi:

$$C_{5t} = f_{11}p$$
$$C_{4t} = C_5 f_{11}p = C_5 c5t$$
$$C_{3t} = C_4 C_5 f_{11}p = C_4 c_{4t}$$
$$C_{2t} = C_3 C_4 C_5 f_{11}p = C_3 c_{3t}$$
$$C_{1t} = C_2 C_3 C_4 C_5 f_{11}p = C_2 c_{2t}$$
$$C_{0t} = C_1 C_2 C_3 C_4 C_5 f_{11}p = C_1 c1t$$

La terza operazione, quella inerente all'azzeramento del registro C, viene eseguita allo stato $f_{15}\bar{G}$

La tavola della Verità corrispondente è uguale a quella di Fig. 4.5.9.
Le equazioni di input sono:

$$c_{it} = C_1 f_{15} \bar{G} p$$

In definitiva, le equazioni di input dei flip-flop relativi al memory-address register sono:

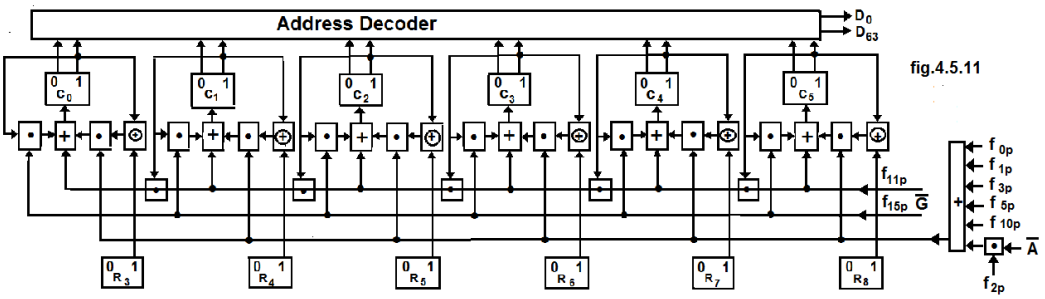
$$\begin{aligned}
c_{0t} &= C_1 C_2 C_3 C_4 C_5 f_{11} p + (R_3 \oplus C_0) g p + C_0 f_{15} \bar{G} p \\
c_{1t} &= C_2 C_3 C_4 C_5 f_{11} p + (R_4 \oplus C_1) g p + C_1 f_{15} \bar{G} p \\
c_{2t} &= C_3 C_4 C_5 f_{11} p + (R_5 \oplus C_2) g p + C_2 f_{15} \bar{G} p \\
c_{3t} &= C_4 C_5 f_{11} p + (R_6 \oplus C_3) g p + C_3 f_{15} \bar{G} p \\
c_{4t} &= C_5 f_{11} p + (R_7 \oplus C_4) g p + C_4 f_{15} \bar{G} p \\
c_{5t} &= f_{11} p + (R_8 \oplus C_5) g p + C_5 f_{15} \bar{G} p
\end{aligned}$$

dove $g = f_{10} + f_0 + f_1 + f_2 \bar{A}_0 + f_3 + f_5$

In fig. 4.5.11 compare un diagramma logico relativo al memory-address register. In questa figura è indicata anche l'address decoder, che non è altro che una matrice con 6 coppie di input e 64 output.

Gli output D_j (j$^{\text{sima}}$ parola) del decodificatore sono rappresentati come:

$$\begin{aligned}
D_0 &= \bar{C}_0 \bar{C}_1 \bar{C}_2 \bar{C}_3 \bar{C}_4 \bar{C}_5 \\
D_1 &= \bar{C}_0 \bar{C}_1 \bar{C}_2 \bar{C}_3 \bar{C}_4 C_5 \\
&\cdots \\
D_{63} &= C_0 C_1 C_2 C_3 C_4 C_5 \qquad (4.5.12)
\end{aligned}$$



12.5.4 Instruction register

L'instruction register permette la memorizzazione temporanea di una parola presa dall'unità di memoria:

Come è mostrato in fig-4.4.5, sono richieste le esecuzioni delle seguenti operazioni:

$$\begin{aligned}
f_{10} &: (C) \Rightarrow Ad[R] \\
f_0, f_1, f_4 &: (M < C >) \Rightarrow R
\end{aligned}$$

La prima operazione, relativa al trasferimento del contenuto del registro C nella parte indirizzo del registro R ha la tabella della verità affiancata.

Le equazioni di input saranno quindi:

$$r_{it} = (C_{i-3} \oplus R_i) f_{10} p \qquad i = 3, ..., 8$$

La seconda operazione prende una parola dalla memoria il cui indirizzo è dato dal contenuto del registro C e la conserva nel registro R.

L'indirizzo contenuto viene indicato con la notazione D_j, dove J specifica la j$^{\text{sima}}$ parola.

Nella memoria del nostro ipotetico calcolatore vi sono 576 bits conservati in 64 parole; essi vengono espressi con il simbolo M_{ji} il quale indica il contenuto del i$^{\text{simo}}$ bit relativo alla j$^{\text{sima}}$ parola.

L'operazione di trasferimento del bit M_{ji} dalla memoria all'i$^{\text{simo}}$ bit del registro Rè la stesssa di quella di tabella 4.5.13, dove C_{i-3} viene sostituito da M_{ji}

Le equazioni di input sono:

$$r_{it} = (\bar{M}_{ji} R_i + M_{ji} \bar{R}_i)(f_0 + f_1 + f_4) p$$

Le equazioni precedenti non sono complete in quanto contemplano solo il trasferimento del contenuto del bit M_{ji} e non considerano la selezione della voce di indirizzo D_j Se il bit i$^{\text{simo}}$ della j$^{\text{sima}}$ parola in memoria viene selezionato mediante il comando D_j (il quale può esseree uno qualsiasi dei 64 comandi di indirizzo) M_{ji} e $\bar{M}_{ji}$ vengono sostituiti dalle equazioni seguenti:

$$\sum_{j=0}^{63} M_{ji} D_j = M_{0i} D_0 + ... + M_{63i} D_{63}$$
$$\sum_{j=0}^{63} \bar{M}_{ji} D_j = M_{0i} D_0 + ... + M_{63i} D_{63}$$

Per cui, le equazioni di input diventano:

$$r_{it} = [\bar{R}_i (\sum_{j=0}^{63} M_{ji} D_j) + R_i (\sum_{j=0}^{63} \bar{M}_{ji} D_j)](f_0 + f_1 + f_4) p \qquad i = 0, ..., 8$$

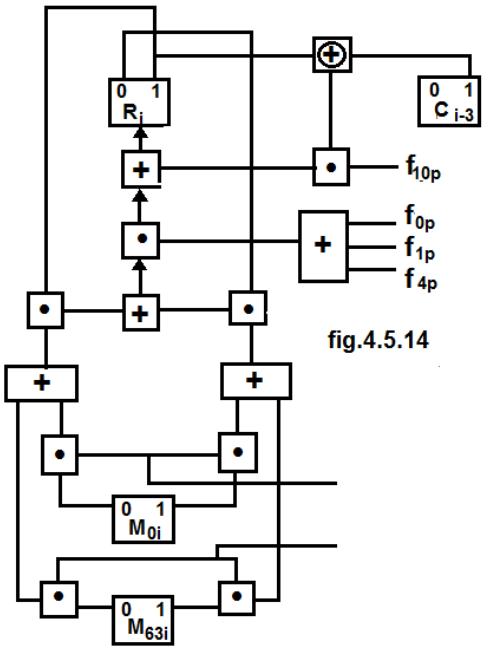
In definitiva, le equazioni di input del registro R saranno:

$$r_{it} = [\bar{R}_i (\sum_{j=0}^{63} M_{ji} D_j) + R_i (\sum_{j=0}^{63} \bar{M}_{ji} D_j)](f_0 + f_1 + f_4) p \qquad i = 0, ..., 2$$

$$r_{it} = [\bar{R}_i (\sum_{j=0}^{63} M_{ji} D_j) + R_i (\sum_{j=0}^{63} \bar{M}_{ji} D_j)](f_0 + f_1 + f_4) p + (C_{i-3} \oplus R_i) f_{10} \qquad i = 3, ..., 8$$

dove D_j è dato dall'equazione (4.5.12).

In fig-4.5.14, compare il diagramma logico dello i$^{\text{simo}}$ stadio del registro R.



12.5.5 Unità di memoria

La memoria nel nostro ipotetico calcolatore è costituita da un insieme di 576 flip-flop: essa deve poter eseguire le due operazioni seguenti:

$$f_3 : (A) \Rightarrow M < C >$$
$$\text{manuale: } (Q_{ji} \Rightarrow M_{ji}$$

La prima operazione memorizza il contenuto dell'accumulatore nella memoria con l'indirizzo selezionato da D_j allo stato f_3"':

$$f_3 D_j : (A_i) \Rightarrow M_{ji} \qquad i = 0, ..., 8$$

Le equazioni di input saranno allora:

$$m_{jit} = (A_i \oplus M_{ji}) f_3 D_j p \qquad i = o, .., 8 \qquad j = 0, ..., 63$$

Q_{ji}	M_{ji}	$M_{ji}(\tau)$	m_{ji}
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

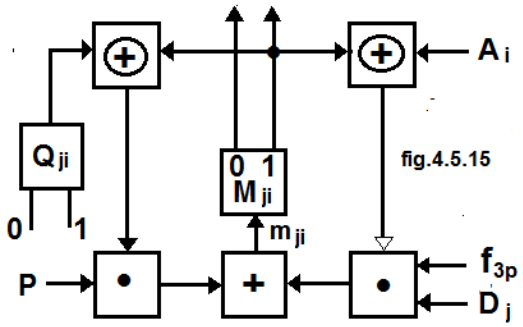
La seconda operazione consiste nell'inserire delle parole in memoria mediante gli interruttori Q

Se Q_{ji} indica l'interruttore connesso al bit di memoria M_{ji}, l'operazione ha la tabella di verità affiancata.

La forma finale delle equazioni di input della unità di memoria è:

$$m_{ji} = (A_i \oplus M_{ji}) f_3 D_j p + (Q_{ji} \oplus M_{ji} p \qquad i = 0, ..., 8 \qquad j = o, ..., 63$$

In fig-4.5.15 compare un diagramma logico abbreviato del registro di memoria.



12.5.6 Start-stop Flip-Flop

La partenza e l'arresto del calcolatore vengono controllati mediante il flip-flop G, il quale viene a sua volta attivato dall'interruttore S mediante operazione manuale.

Le due operazioni richieste sono:

$$f_5 C_3 : 0 \Rightarrow G$$
$$\text{manuale: } (S) \Rightarrow G$$

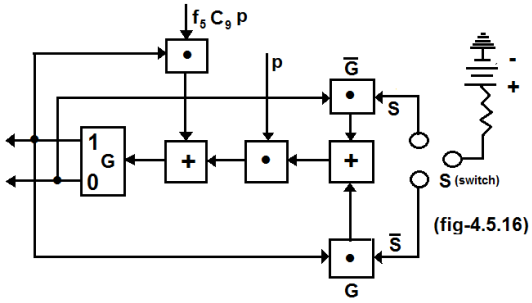
La prima operazione cambia il Flip-Flop G nello stato 0 quando compare lo stato f_5 e C_3 è 1; le equazioni di input sono quindi:

$$g_t = G f_5 C_3 p$$

La seconda operazione mette in **ON** od in **OFF** l'interruttore S per permettere l'avvio o l'arresto della macchina.

L'equazione di input è:

$$g_t = G f_5 C_3 p + (S \oplus G 9 p$$



12.6 Un calcolatore completo

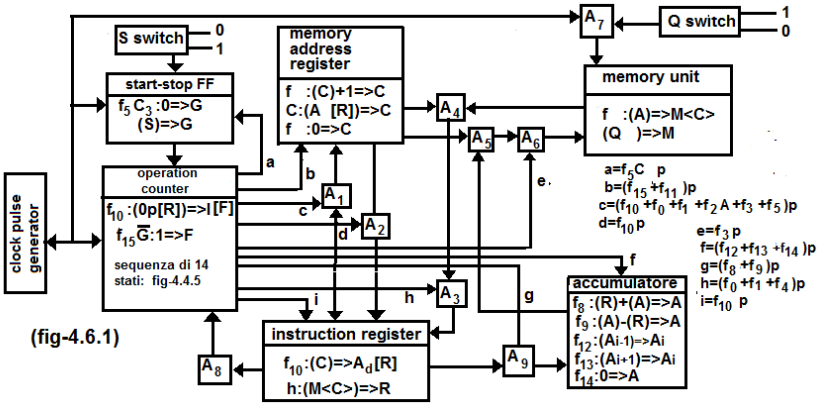
Nei paragrafi precedenti abbiamo stabilito le equazioni di **input** di tutti i registri del nostro ipotetico calcolatore.

La configurazione del calcolatore dato in fig-4.3.1 può essere ora completata dando le equazioni simboliche relative ai singoli registri (fig. 4.6.1).

Questo diagramma può essere reso più dettagliato usando i diagrammi logici stabiliti nei precedenti paragrafi.

La procedura operativa del nostro calcolatore sarà la seguente:

1. Accensione del calcolatore. Se il Flip-Flop di avvio-arresto si trova nello stato 1 durante questo periodo, il calcolatore funzioneera in maniera irregolare.
2. Si mette lo switch S nella posizione **off**. Questo implica che lo start-stop Flip-Flop sia nello stato 0, lo operation counter in 1111 e l'indirizzo contenuto nel memory address register uguale a 000000. Si inserisce il programma in memoria mediante gli interruttori Q.
3. Si mette lo switch S nella posizione **ON**. Il Flip-Flop Gassume lo stati 1 e la macchina comincia ad esguire il programma. La prima istruzione si trova nella memoria di indirizzo 000000.
4. Il programma deve terminare con una istruzione di stop: questa ultima inserisce 0 in G, 1111 in F e 000000 in C. Un nuovo programma può essere inserito nella memoria ed il calcolatore può iniziare il calcolo rimettendo lo switch nella posizione **ON**.
5. Si leggono i risultati ottenuti mediante dispositivi luminosi connessi ai flip-flop di memoria (non mostrati nei diagrammi precedenti).



Fonte del testo:
https://it.wikibooks.org/w/index.php?title=Algebre_booleane_e_progetto_logico_dei_calcolatori_digitali/Progetto_logico_di_un_calcolatore_digitale&oldid=402736

Crediti

Indice delle immagini

Tutte le immagini sono di Alfonso Sommacal e rilasciate con licenza Creative Commons Attribuzione-Condividi allo stesso modo 4.0 Internazionale.

- p. 94 - *fonte:* https://commons.wikimedia.org/wiki/File:Funzione_booleana.png
- p. 94 - *fonte:* https://commons.wikimedia.org/wiki/File:Circuiti_logici_equivalenti.png
- p. 95 - *fonte:* https://commons.wikimedia.org/wiki/File:Circuiti_logici_equivalenti_(3)_e_(4).png
- p. 96 - *fonte:* https://commons.wikimedia.org/wiki/File:Funzione_sequenziale_elementare.png
- p. 97 - *fonte:* https://commons.wikimedia.org/wiki/File:Crcuito_chiusi_o_ad_anello.png
- p. 97 - *fonte:* https://commons.wikimedia.org/wiki/File:Operatore_memoria_elementare.png
- p. 99 - *fonte:* https://commons.wikimedia.org/wiki/File:Operatore_memoria_elementare.png
- p. 100 - *fonte:* https://commons.wikimedia.org/wiki/File:Memoria_dinamica-Flip-Flop_S-R.png
- p. 100 - *fonte:* https://commons.wikimedia.org/wiki/File:Memoria_dinamica-Flip-Flop_T.png
- p. 100 - *fonte:* https://commons.wikimedia.org/wiki/File:Esempio_sequenza_di_ingresso_e_di_uscita_per_un_Flip-Flop.png
- p. 101 - *fonte:* https://commons.wikimedia.org/wiki/File:Ritardo_di_sequenza.png
- p. 101 - *fonte:* https://commons.wikimedia.org/wiki/File:Circuito_di_ritardo_dinamico.png
- p. 102 - *fonte:* https://commons.wikimedia.org/wiki/File:Grafico_i_funzione_di_V.png
- p. 102 - *fonte:* https://commons.wikimedia.org/wiki/File:Comportamento_di_un_diodo_ideale.png
- p. 102 - *fonte:* https://commons.wikimedia.org/wiki/File:Porta_a_diodi.png
- p. 103 - *fonte:* https://commons.wikimedia.org/wiki/File:Porta_a_diodi_funzione_F2.png
- p. 103 - *fonte:* https://commons.wikimedia.org/wiki/File:Comportamento_ideale_di_un_transistor.png
- p. 103 - *fonte:* https://commons.wikimedia.org/wiki/File:Transistor_NPN.png
- p. 104 - *fonte:* https://commons.wikimedia.org/wiki/File:Comportamento_ideale_di_un_transistor.png
- p. 104 - *fonte:* https://commons.wikimedia.org/wiki/File:Caratteristica_operativa_di_un_transistore.png
- p. 105 - *fonte:* https://commons.wikimedia.org/wiki/File:Segnali_invertiti_e_tabella_della_verit%C3%A0.png
- p. 105 - *fonte:* https://commons.wikimedia.org/wiki/File:Cifrcuiti_NAND_e_NOR.png
- p. 106 - *fonte:* https://commons.wikimedia.org/wiki/File:Simboli_dei_circuiti_NAND_e_NOR.png
- p. 106 - *fonte:* https://commons.wikimedia.org/wiki/File:Flip-Flop_logic_circuit..png
- p. 107 - *fonte:* https://commons.wikimedia.org/wiki/File:Simplified_circuit_diagram_of_Flip-Flop_S-R.png

- p. 135 - *fonte:* https://commons.wikimedia.org/wiki/File:Decade_per_codici_ad_eccesso_di_tre.png
- p. 137 - *fonte:* https://commons.wikimedia.org/wiki/File:Contatore_in_binario_riflesso.png
- p. 137 - *fonte:* https://commons.wikimedia.org/wiki/File:Contatore_in_binario_riflesso_(bis).png
- p. 138 - *fonte:* https://commons.wikimedia.org/wiki/File:Diagramma_di_stato.png
- p. 138 - *fonte:* https://commons.wikimedia.org/wiki/File:Diagamma_di_stato_a_vertici_rappresentativi.png
- p. 138 - *fonte:* https://commons.wikimedia.org/wiki/File:Grafo_di_stato_2.10.1_a.png
- p. 139 - *fonte:* https://commons.wikimedia.org/wiki/File:Diagramma_di_stato_di_un_contatore_decimale.png
- p. 141 - *fonte:* https://commons.wikimedia.org/wiki/File:Rappresentazione_simbolica_di_un_registro_a_scorrimento.png
- p. 141 - *fonte:* https://commons.wikimedia.org/wiki/File:Registri_a_scorrimento.png
- p. 142 - *fonte:* https://commons.wikimedia.org/wiki/File:Registro_a_scorrimento_costituito_da_Flip-Flop.png
- p. 144 - *fonte:* https://commons.wikimedia.org/wiki/File:Trasferimento_dal_registo_A_al_registo_B.png
- p. 145 - *fonte:* https://commons.wikimedia.org/wiki/File:Diagramma_logico_addizonatgore_seriale_binario.png
- p. 146 - *fonte:* https://commons.wikimedia.org/wiki/File:Diagramma_logico_di_un_accumulatore_seriale.png
- p. 146 - *fonte:* https://commons.wikimedia.org/wiki/File:Diagramma_logico_di_un_accumulatore_binario_parallelo..png
- p. 147 - *fonte:* https://commons.wikimedia.org/wiki/File:Diagramma_di_accumulatore_binario_parallelo_(addizione).png
- p. 147 - *fonte:* https://commons.wikimedia.org/wiki/File:Diagramma_di_accumulatore_binario_parallelo_sottrattore.png
- p. 148 - *fonte:* https://commons.wikimedia.org/wiki/File:Accumulatore_addizionatore-sottrattore.png
- p. 149 - *fonte:* https://commons.wikimedia.org/wiki/File:Matrice_booleana.png
- p. 151 - *fonte:* https://commons.wikimedia.org/wiki/File:Matrice_rettangolare_utilizzante_circuito_AND.png
- p. 152 - *fonte:* https://commons.wikimedia.org/wiki/File:Forma_matriciale_di_un_circuito.png
- p. 153 - *fonte:* https://commons.wikimedia.org/wiki/File:Forma_logica_di_una_matrice_di_commutazione.png
- p. 153 - *fonte:* https://commons.wikimedia.org/wiki/File:Matrice_di_commutazione_adf_albero_con_circuiti_AND.png
- p. 154 - *fonte:* https://commons.wikimedia.org/wiki/File:Matrice_di_commutazione_a_doppio_albeero_con_circuiti_AND.png
- p. 155 - *fonte:* https://commons.wikimedia.org/wiki/File:Matrici_a_doppio_albero_con_otto_coppie_di_inputs_e_256_outputs.png
- p. 156 - *fonte:* https://commons.wikimedia.org/wiki/File:Decodifcatore.png

- p. 187 - *fonte:* https://commons.wikimedia.org/wiki/File:Figure_4.5.16.png
- p. 188 - *fonte:* https://commons.wikimedia.org/wiki/File:Operazioni_in_un_calcolatore_digitale_semplificato.png