CUD

Consorzio per l'Università a Distanza

Calcolatori Elettronici

Architettura al livello logico - digitale

*

Diploma Universitario in Ingegneria Informatica e Automatica

Architettura al livello logico - digitale

Bruno

CONSORZIO PER L'UNIVERSITÀ A DISTANZA

CALCOLATORI ELETTRONICI

1. RAPPRESENTAZIONE DELLE INFORMAZIONI
2. ALGEBRA BOOLEANA
3. INTRODUZIONE ALL'ARCHITETTURA

RENDE 1993

DIPLOMA UNIVERSITARIO IN INGEGNERIA INFORMATICA E AUTOMATICA

Diploma Universitario in Ingegneria Informatica e Automatica

Modulo: Architettura al livello logico-digitale

Autore: G. Cioffi

Progettazione didattica ed editing a cura del CUD

luglio 1993

V. Marconi 32, 87030 Rende (CS)
Seconda edizione

ISBN 88-7721-224-1

INDICE

Unità 1 Rappresentazione delle informazioni

INDICE

Unità 2 Algebra Booleana

INDICE

Unità 3 Introduzione all'architettura dei calcolatori

Unità 1

Rappresentazione delle informazioni

PREMESSA

Questa unità costituisce, insieme con la seguente, la parte introduttiva del corso, che mira a fornire le basi teoriche per la comprensione degli argomenti trattati in tutte le restanti unità.
La prima unità è dedicata, in generale, alla rappresentazione delle informazioni e descrive la forma in cui possono essere rappresentati numeri e caratteri per essere immagazzinati ed elaborati all'interno di un calcolatore elettronico. Tale forma è molto diversa da quella cui siamo quotidianamente abituati; per bene assimilare i contenuti del corso è necessario comprenderla a fondo e saperla usare con naturalezza.
La seconda unità introduce alcuni importanti algoritmi applicabili ai simboli con cui si rappresentano le informazioni nei calcolatore. Questi algoritmi sono interpretabili anche come la descrizione formale del funzionamento di alcuni circuiti elettronici elementari (le porte logiche) che compongono materialmente i calcolatori. Prima di iniziare il nostro studio, vogliamo avvertirti che, per la natura e il numero degli argomenti trattati, queste due prime unità sono senza dubbio le più aride e pesanti del corso. L'aver raggruppato tutte le nozioni propedeutiche ci consentirà tuttavia di procedere speditamente e più direttamente nelle successive unità. Con questa motivata certezza, ti auguriamo un buono studio.

0. INTRODUZIONE

0.1. Presentazione

Questa unità è dedicata alla rappresentazione delle informazioni, cioè come le "informazioni" che vogliamo trattare possono venire espresse in forma comprensibile al calcolatore.
Tralasciando le implicazioni filosofiche insite nel termine, cominceremo coll'osservare che qualsiasi "informazione" scritta può essere espressa con un insieme molto limitato di simboli: le lettere dell'alfabeto, le cifre decimali, un certo gruppo di caratteri speciali (? & $; . ecc.): in tutto un centinaio di simboli diversi, chiamati caratteri.
Il numero di caratteri – pur essendo relativamente piccolo – è molto più grande delle capacità interpretative dei circuiti di un calcolatore. Questi circuiti,infatti, sanno riconoscere due soli simboli, convenzionalmente indicati con 0 e 1, che costituiscono l'unità elementare di informazione, e sono noti come già sappiamo, col nome di bit (binary digit = cifra binaria).
Come vedremo dettagliatamente in seguito:

1) i bit non hanno necessariamente un significato numerico;
2) nei circuiti dei calcolatori lo 0 e l'1 corrispondono fisicamente a due diversi e ben definiti valori di tensione (ad esempio, 0. *V* e +5, *V*).

Con l'unità elementare di informazione, o bit, è possibile quindi esprimere informazioni complesse. In termini più espliciti: è possibile trovare un modo per rappresentare – con i soli simboli 0 e 1 – tutti i caratteri con cui esprimiamo per iscritto qualsiasi informazione. Un esempio banale di questa possibilità è data dai pannelli luminosi dove, combinando opportunamente lampadine accese e spente che "materializzano" i simboli 1 e 0, è possibile formare qualsiasi scritta.
La possibilità di ridurre il numero di simboli usati per esprimere un'informazione è un problema di non difficile soluzione. Ad esempio, se, per convenzione, facciamo corrispondere ognuno dei primi cento numeri interi ai caratteri di cui ci serviamo per scrivere le informazioni, possiamo rappresentare le informazioni stesse con le sole dieci cifre decimali con cui scriviamo normalmente qualsiasi numero.
Se, a questo punto, troviamo poi un modo per rappresentare ogni cifra decimale con i soli simboli 0 e 1, abbiamo ottenuto un modo per rappresentare ogni informazione con stringhe (cioè successioni) di soli bit.
In questa unità vedremo prima come rappresentare i numeri con stringhe di bit (sistema binario), e come considerare il sistema binario nell'ambito dei sistemi di numerazione posizionali. Vedremo poi come vengono eseguite le operazioni aritmetiche direttamente nel sistema binario.
Esamineremo, infine, i più comuni codici, cioè le regole per rappresentare i simboli della scrittura mediante stringhe di bit.

0.2. Prerequisiti

Per lo studio di questa unità non sono necessari prerequisiti specifici.

0.3. Obiettivi

Alla fine dello studio di questa unità conoscerai il modo di rappresentare le informazioni all'interno dei sistemi di calcolo, sia che si tratti di informazioni numeriche, sia che si tratti di altri tipi di informazioni.

0.4. Parole chiave

In questa unità incontrerai molti termini nuovi. Essi sono, in ordine di apparizione, i seguenti:

sistema decimale	1.1.
base	1.2.
sistema binario	1.3.
sistema ottale	1.4.
sistema esadecimale	1.5.
complemento alla base	1.7.
rappresentazione in virgola mobile	1.8.
codice	3.1.
codice binario ridondante	3.2
codice binario irridondante	3.2.
distanza di Hamming	3.2.
errore	3.2.
codice di BCD	3.3.
codice di Gray	3.3.
codice ASCII	3.3.
codice di parità	3.4.

1. SISTEMI NUMERICI

1.1. Introduzione

Usualmente siamo abituati a rappresentare i numeri nel sistema numerico decimale, utilizzando le dieci cifre da 0 a 9, e dando a ogni cifra un peso diverso, a seconda della sua posizione all'interno del numero. Ad esempio, la scrittura 653,32 va interpretata come la rappresentazione del numero il cui valore è:

$$6 * 100 + 5 * 10 + 3 * 1 + 3 * 0,1 + 2 * 0,01$$

cioè:

$$6 * 10^2 + 5 * 10^1 + 3 * 10^0 + 3 * 10^{-1} + 2 * 10^{-2}.$$

Questa rappresentazione dei numeri è chiamata anche "posizionale in base 10", perché il coefficente (peso) per cui va moltiplicata ogni cifra è una potenza della base 10, potenza il cui esponente dipende dalla posizione della cifra rispetto a un punto di riferimento che è la virgola. Nulla vieta tuttavia di rappresentare i numeri anche in una base diversa da 10, per esempio 2 (sistema binario), 8 (sistema ottale) o 16 (sistema esadecimale).

1.2. Sistemi di numerazione posizionali

Per ogni sistema di numerazione posizionale, valgono le seguenti proprietà:

- la base (b), o "radice", coincide col numero totale di cifre usate dal sistema; notiamo esplicitamente che deve essere $b > 1$;
- la cifra di minor valore è 0; le altre sono, $1, 2, \ldots, b-1$;
- se è $b > 10$, si introducono ulteriori $b - 10$ cifre, indicandole con $A, B, C \ldots$;
- un numero è una stringa di cifre, contenente o meno la virgola decimale; la cifra all'estrema destra ha il valore minore (cifra meno significativa), quella all'estrema sinistra ha il valore maggiore (cifra più significativa).

Indicando con c_i la cifra nella generica posizione i, la rappresentazione di un numero N è una stringa di cifre $(N)_b = c_s c_{s-1} \ldots c_1 c_0, c_{-1} c_{-2} \ldots c_{-r}$ di lunghezza $s + r + 1$.
Il valore del numero N, indicato con $V(N)$, è dato dal polinomio seguente:

$$V(N) = c_s b^s + c_{s-1} b^{s-1} + \cdots + c_1 b^1 + c_0 b^0 + c_{-1} b^{-1} + c_{-2} b^{-2} + \cdots + c_{-r} b^{-r}$$

essendo $V(c_i)$ compreso tra 0 e $b - 1$.
Come si è già detto i sistemi di numerazione così definiti sono chiamati posizionali perché la generica cifra c_i è moltiplicata per un peso il cui valore dipende dalla posizione della cifra nella stringa. In generale il numero N è costituito da una parte intera $I = c_s c_{s-1} \ldots c_1 c_0$ e da una parte frazionaria $F = c_{-1} c_{-2} \ldots c_{-r}$.

Esempio 1. Il numero $(325,23)_{10}$ ha il valore:

$$3 * 10^2 + 2 * 10^1 + 5 * 10^0 + 2 * 10^{-1} + 3 * 10^{-2}$$

mentre il numero $(325,23)_8$ ha il valore:

$$3 * 8^2 + 2 * 8^1 + 5 * 8^0 + 2 * 8^{-1} + 3 * 8^{-2}$$

L'esempio 1 mostra la necessità di indicare la base quando si ha a che fare con sistemi di numerazione a base diversa. Nei numeri che siamo abituati a vedere quotidianamente, la base è omessa in quanto si suppone tacitamente che essa sia 10. La soppressione della base è ugualmente permessa ogni volta che non esista possibilità di equivoco.

Domanda 1. Con quale scrittura viene espressa in ogni sistema di numerazione posizionale il valore della base b del sistema di numerazione?

Nei prossimi paragrafi di questa sezione, esamineremo le proprietà generali dei sistemi numerici posizionali, nonché quelle particolari dei sistemi binario, ottale ed esadecimale. Vedremo quindi come vengono effettuate le conversioni dall'uno all'altro dei sistemi numerici predetti. Esamineremo infine le rappresentazioni binarie dei numeri negativi, e quelle in "virgola mobile".

1.3. Il sistema binario

Come già detto, le due cifre del sistema binario (0 e 1) si chiamano bit. Un numero binario è dunque una stringa di bit. Il suo valore è la somma delle potenze di 2 in corrispondenza delle posizioni in cui compare il bit 1.

Esempio 2. Il numero binario 1011,001 ha il valore:

$$1*2^3+0*2^2+1*2^1+1*2^0+0*2^{-1}+0*2^{-2}+1*2^{-3} = 8+2+1+0,125 = 11,125.$$

Osservazione importante: nell'esempio 2, il valore del numero binario è in realtà espresso in decimale. Con maggior precisione, avremmo dovuto scrivere:

$$(1011,001)_2 = (2^3)_{10} + (2^1)_{10} + (2^0)_{10} + (2^{-3})_{10} = (11,125)_{10}.$$

Quest'ultima scrittura è una uguaglianza fra espressioni in due sistemi di numerazione diversi. L'uguaglianza esprime infatti una "conversione di base" dal sistema binario al sistema decimale (l'argomento sarà ripreso nel paragrafo 1.6.)

Esercizio 1. Prova a scrivere il valore del numero binario 1011.001 nel sistema di numerazione decimale.

Domanda 2. Sapresti dire come si fa a riconosce un numero binario dispari?

È interessante stabilire quante cifre occorrono per rappresentare in binario "qualsiasi" numero decimale di n cifre. Nel sistema decimale, con n cifre, possiamo rappresentare tutti i numeri compresi tra 0 e $10^n - 1$ (considerando, per semplicità, i soli numeri interi). Ad esempio, con 5 cifre possiamo rappresentare tutti i numeri tra 0 e 99999.

Analogamente, nel sistema binario, possiamo rappresentare, con m bit, tutti gli interi compresi tra 0 e $2^m - 1$. Pertanto, il numero m di bit occorrenti per trasformare in binario "qualsiasi" numero decimale di n cifre si ricava dalla disequazione:

$$2^m - 1 \geq 10^n - 1$$

che, risolta, dà:

$$m \geq n * \log_2 10 \geq 3,32 * n.$$

Tenendo conto che m deve essere un numero intero la relazione può essere scritta in modo più corretto nel modo seguente:

$$m = \lceil n * \log_2 10 \rceil.$$

In pratica, per n sufficientemente grande, il numero di bit necessari per rappresentare "qualsiasi" numero in binario è di poco superiore al triplo delle cifre necessarie per rappresentarlo in decimale.
Nota che abbiamo sempre messo in evidenza la parola "qualsiasi": in effetti, le cifre binarie e le cifre decimali necessarie per rappresentare un numero dipendono dal valore del numero stesso (così, ai numeri decimali 1, 2 e 8 corrispondono i numeri binari 1, 10 e 100). Per essere sicuri di poter rappresentare qualsiasi numero decimale di n cifre, ci siamo pertanto riferiti al valore massimo $10^n - 1$.

Esempio 3. Per rappresentare il numero decimale 9999 occorrono, secondo la relazione $m \geq 3,32 * n$, 14 cifre (14 è il più piccolo intero maggiore del valore 13,28). In effetti il numero $(9999)_{10}$ corrisponde al binario 10011100001111.

Esercizio 2. Prova a calcolare il numero di cifre necessarie per rappresentare un numero decimale, supponendo che, nel sistema binario, il numero sia rappresentato con una stringa qualsiasi di m bit. Calcola, in particolare, il numero di cifre decimali occorrenti per rappresentare il più grande intero di 64 bit.

1.4. Il sistema ottale

Il sistema ottale è il sistema numerico a base 8; pertanto usa le otto cifre dallo 0 al 7. Un numero ottale è una stringa di cifre ottali; indicando con c_i la cifra ottale nella posizione i, la rappresentazione di un numero N in base ottale ha quindi la forma:

$$(N)_8 = c_s c_{s-1} \dots c_1 c_0, c_{-1} c_{-2} \dots c_{-r}$$

il valore decimale è dato dal polinomio:

$$V(N) = c_s * 8^s + c_{s-1} * 8^{s-1} + c_1 * 8^1 + c_0 * 8^0 + c_{-1} * 8^{-1} + c_{-2} * 8^{-2} + \dots + c_{-r} * 8^{-r}$$

in accordo con la formula generale riportata nel paragrafo 1.2.

Esempio 4. L'espressione decimale del numero ottale 166,702 è:

$$1*8^2 + 6*8^1 + 6*8^0 + 7*8^{-1} + 0*8^{-2} + 2*8^{-3} = 64 + 48 + 6 + \frac{7}{8} + \frac{2}{8^3}$$

cioè: $(166,702)_8 = (118,8789)_{10}$.

Nella letteratura specializzata si legge spesso che un calcolatore "usa la rappresentazione ottale": questo modo di dire può portare a confusioni. In effetti il calcolatore usa sempre e soltanto la rappresentazione binaria; per comodità di scrittura, tuttavia, si rappresentano spesso le stringhe di bit in stringhe di cifre ottali, che sono tre volte più corte. Le regole di trasformazione dal sistema binario a quello ottale e viceversa sono esposte nel paragrafo 1.6.

Domanda 3. Qual è l'espressione decimale del numero ottale 70654?

1.5. Il sistema esadecimale

Il sistema esadecimale è il sistema numerico con base 16. Usa le cifre da 0 a 9 e le lettere A, B, C, D, E, F col valore, rispettivamente, dei numeri decimali 10, 11, 12, 13, 14, 15. Un numero esadecimale è una stringa di cifre esadecimali; indicando con c_i la cifra esadecimale nella posizione i, la rappresentazione di un numero N in base esadecimale ha la forma:

$$(N)_{16} = c_s c_{s-1} \dots c_1 c_0, c_{-1} c_{-2} \dots c_{-r}$$

ed il valore decimale dato dal polinomio:

$$V(N) = c_s * 16^s + c_{s-1} * 16^{s-1} + \dots + c_1 * 16^1 + c_0 * 16^0 + \\ + c_{-1} * 16^{-1} + c_{-2} * 16^{-2} + \dots + c_{-r} * 16^{-r}$$

in accordo con la formula generale del paragrafo 1.2.

Esempio 5. L'espressione decimale del numero esadecimale A2,34 è:

$$A * 16^1 + 2 * 16^0 + 3 * 16^{-1} + 4 * 16^{-2} = 10 * 16 + 2 + \frac{3}{16} + \frac{4}{256}$$

cioè:

$$(A2,34)_{16} = (162,203)_{10}.$$

Come abbiamo appena visto per il sistema ottale, l'espressione "calcolatore che usa la rappresentazione esadecimale" va interpretata nel senso che le stringhe di bit vengono rappresentate in stringhe di cifre esadecimali, che sono quattro volte più corte (vedi il paragrafo 1.6.).

Domanda 4. Saprésti trovare l'espressione decimale del numero esadecimale CAFFE?

1.6. Conversione di base

Dovendo adoperare più sistemi di numerazione, è necessario saper esprimere i numeri in ognuno di essi, trasformandoli mediante opportune regole per la conversione di base.

Il problema di convertire un numero N da una base b ad una base a è affrontato in modo separato per la parte intera e per quella frazionaria. Per la parte intera I, conviene riscrivere l'espressione polinomiale in forma diversa:

$$I = c_0 + a * (c_1 + a * (c_2 + a(c_3 + a * (+ \cdots + a * c_s)))) \ldots)$$

il problema consiste nel trovare le $s+1$ cifre c_i nella rappresentazione di N in base a. Scriviamo ora I nella forma: $I = a * Q + R$ $(0 < R < a)$ dove Q è il quoziente intero di I/a, ed R è il resto.
Uguagliando le due espressioni di I otteniamo:

$$R + a * Q = c_0 + a * (c_1 + a * (+ \cdots + a * c_s)) \ldots)$$

e cioè:

$$c_0 = R$$

$$c_1 + a * (c_2 + a * (\ldots) \ldots) = Q.$$

Se ora facciamo assumere a Q il ruolo di I, possiamo ripetere il procedimento ricavando c_1 come resto R di Q/a...e così via. La conversione termina quando otteniamo un quoziente nullo.

Esempio 6. Trasformiamo il numero decimale 325 in binario:

$$\begin{array}{ll} 325 : 2 = 162 + 1; & c_0 = 1 \\ 162 : 2 = 81 + 0; & c_1 = 0 \\ 81 : 2 = 40 + 1; & c_2 = 1 \\ 40 : 2 = 20 + 0; & c_3 = 0 \\ 20 : 2 = 10 + 0; & c_4 = 0 \\ 10 : 2 = 5 + 0; & c_5 = 0 \\ 5 : 2 = 2 + 1; & c_6 = 1 \\ 2 : 2 = 1 + 0; & c_7 = 0 \\ 1 : 2 = 0 + 1; & c_8 = 1 \end{array}$$

Si ha quindi: $(325)_{10} = (101000101)_2$.

Per la parte frazionaria F si procede in modo inverso. Posto:

$$F = \frac{1}{a} * \left(c_{-1} + \frac{1}{a} * \left(c_{-2} + \frac{1}{a} * \left(\ldots \frac{1}{a} * c_{-r}\right)\right) \ldots\right)$$

e:

$$F * a = I' + F'$$

si ottiene:

$$I' + F' = \left(c_{-1} + \frac{1}{a} * \left(c_{-2} + \frac{1}{a} * \left(\ldots \frac{1}{a} * c_{-r}\right)\right) \ldots\right)$$

e cioè:

$$c_{-1} = I'$$

$$\frac{1}{a} * \left(c_{-2} + \frac{1}{a} * (\ldots) \ldots\right) = F'.$$

Se applichiamo lo stesso procedimento a F', possiamo ricavare c_{-2} e così via, fino a quando saremo arrivati a una parte frazionaria nulla, o avremo raggiunto la precisione desiderata.
Nota che se arrestiamo il procedimento alla cifra c_{-r}, effettuiamo un troncamento che comporta un errore assoluto minore di a^{-r}.

Esempio 7. Convertiamo in binario il numero decimale 0,775:

$$\begin{array}{llll} 0,7775 * 2 = 1,555; & I' = 1; & c_{-1} = 1; & F' = 0,555 \\ 0,555 * 2 = 1,110; & I' = 1; & c_{-2} = 1; & F' = 0,110 \\ 0,110 * 2 = 0.220; & I' = 0; & c_{-3} = 0; & F' = 0,220 \\ 0,220 * 2 = 0,440; & I' = 0; & c_{-4} = 0; & F' = 0,440 \\ 0,440 * 2 = 0,880; & I' = 0; & c_{-5} = 0; & F' = 0,880 \\ 0,880 * 2 = 1,760; & I' = 1; & c_{-6} = 1; & F' = 0,760 \\ 0,760 * 2 = 1,520; & I' = 1; & c_{-7} = 1; & F' = 0,520 \\ 0,520 * 2 = 1,040; & I' = 1; & c_{-8} = 1; & F' = 0,040 \\ 0,040 * 2 = 0,080; & I' = 0; & c_{-9} = 0; & F' = 0,080 \end{array}$$

Si ha pertanto. $(0,775)_{10} = (0,110001110)_2$.

Domanda 5. Qual è il numero binario corrispondente al numero decimale 4805,0234375 ?

Con i procedimenti esposti, che hanno carattere del tutto generale, possiamo operare conversioni tra due basi qualsiasi. In pratica, tuttavia, hanno importanza soltanto le conversioni tra i sistemi decimale, binario, ottale ed esadecimale. Alcune di queste conversioni vengono effettuate più speditamente con degli speciali accorgimenti, nel modo di seguito illustrato.
Le conversioni dai sistemi binario, ottale ed esadecimale al sistema decimale vengono effettuate esprimendo direttamente in decimale le potenze di 2, 8 e 16, secondo i metodi che abbiamo visto, rispettivamente, nei paragrafi 1.3, 1.4, 1.5.
Le conversioni dal sistema decimale ai sistemi binario, ottale ed esadecimale si effettuano col procedimento esposto in questo stesso paragrafo.
Per fissare le idee, vediamo due esempi di conversione di un numero decimale nei sistemi ottale ed esadecimale.

Esempio 8. Converti in ottale il numero decimale 29100:

$$\begin{array}{ll} \frac{29100}{8} = 3637 + 4 & c_0 = 4 \\ \frac{3637}{8} = 454 + 5 & c_1 = 5 \\ \frac{454}{8} = 56 + 6 & c_2 = 6 \\ \frac{56}{8} = 7 + 0 & c_3 = 0 \\ \frac{7}{8} = 0 + 7 & c_4 = 7 \end{array}$$

pertanto, $(29100)_{10} = (70654)_8$.

Esempio 9. Converti in esadecimale il numero decimale 0.3333:

$$\begin{array}{lll} 0,3333 * 16 = 5,3328; & I' = 5; c_{-1} = 5; & F' = 0,3328 \\ 0,3328 * 16 = 5,3248; & I' = 5; c_{-2} = 5; & F' = 0,3248 \\ 0,3248 * 16 = 5,1968; & I' = 5; c_{-3} = 5; & F' = 0,1968 \\ 0,1968 * 16 = 3,1488; & I' = 3; c_{-4} = 3; & F' = 0,1488 \end{array}$$

pertanto: $(0,3333)_{10} = (0,5533)_{16}$.
Notiamo che la conversione è stata arrestata alla quarta cifra dopo la virgola. Ciò significa, in conformità con quanto detto precedentemente in questo stesso paragrafo, che l'errore di troncamento è minore di 16^{-4}.

Domanda 6. Qual è il corrispondente ottale del numero decimale 6005,6005?

Le conversioni tra i sistemi binario e ottale sono molto semplici, essendo 8 una potenza di 2. Per trasformare quindi un numero binario in ottale, basta raggruppare i bit a tre a tre, partendo dalla virgola, e sostituire ad ogni terna di bit (eventualmente completata con zeri non significativi) la corrispondente cifra ottale.

Esempio 10. Il numero binario 11010101110,1 si scrive dapprima nella forma:

011 010 101 110, 100

e poiché

$$(011)_2 = (3)_8; \quad (010)_2 = (2)_8; \quad (101)_2 = (5)_8; \quad (110)_2 = (6)_8; \quad (100)_2 = (4)_8$$

si ha:

$$(11010101110,1)_2 = (3256,4)_8.$$

Inversamente, per scrivere in binario un numero ottale, basta sostituire ad ogni cifra ottale la corrispondente terna di bit.

Esempio 11. Per il numero $(14560)_8$, si avrà:

$$(1)_8 = (001)_2; \quad (4)_8 = (100)_2; \quad (5)_8 = (101)_2; \quad (6)_8 = (110)_2; \quad (0)_8 = (000)_2.$$

Pertanto il numero si scriverà in binario come 1100101110000, dopo aver eliminato i primi due zeri non significativi.

Allo stesso modo, per rappresentare in esadecimale un numero binario, basta dividere i bit in gruppi di quattro, partendo dalla virgola, e sostituire ad ogni gruppo la corrispondente cifra esadecimale.

Esempio 12. Il numero binario 10110111010010,11 si scrive nella forma: 0010 1101 1101 0010, 1100 e pertanto equivale al numero esadecimale 2DD2,C.

Inversamente, un numero esadecimale si scrive in binario sostituendo, ad ogni cifra esadecimale, il corrispondente gruppo di quattro bit.

Esempio 13. Il numero esadecimale 105F,A si scrive in binario 0001000001011111,1010 o meglio, eliminando gli zeri non significativi, 1000001011111,101.

Anche se non molto di frequente, può essere necessaria una conversione tra i sistemi ottale ed esadecimale. In questi casi, conviene sempre effettuare la conversione passando attraverso il sistema binario.

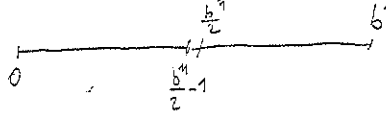
Esempio 14. Il numero ottale 34560 si scrive in binario come 11100101110000 e in esadecimale (dopo aver diviso i bit in gruppi di quattro) come 3970.

Abbiamo visto che la conversione tra il sistema binario e quello ottale, o esadecimale, è estremamente semplice; abbiamo poi detto che "conviene" effettuare la conversione tra i due ultimi sistemi con una conversione intermedia attraverso il sistema binario. Ti proponiamo, a questo punto, di spiegare il perché di questa regola, risolvendo il seguente esercizio.

Esercizio 3. Dimostra che è possibile convertire un numero binario in ottale (esadecimale) convertendo separatamente i gruppi di tre (quattro) bit, e che non è possibile fare lo stesso per la conversione tra i numeri ottali ed esadecimali.

1.7. Rappresentazione dei numeri relativi

Il modo più semplice per rappresentare i numeri relativi è quello in modulo e segno. Per i numeri binari, basta aggiungere un bit di segno s alla stringa che rappresenta il numero, con la convenzione che $s = 0$ indica un segno positivo, ed $s = 1$ indica un segno negativo. Ad esempio, i numeri decimali +20 e −20 si rappresentano, rispettivamente, con 010100 e 110100.
La rappresentazione in modulo e segno non è tuttavia la più comoda per effettuare le operazioni aritmetiche in modo meccanico; si usa, per questo, la rappresentazione in "complemento alla base".
Come abbiamo appena visto, in un sistema a base b, possiamo rappresentare, combinando in tutte le maniere possibili n cifre, tutti i numeri tra 0 e $b^n - 1$. Si chiama complemento alla base b con n cifre la rappresentazione in cui i numeri positivi, lo 0 incluso, sono compresi nell'intervallo $(0, b^n/2 - 1)$ e quelli negativi sono compresi nell'intervallo $(b^n/2, b^n - 1)$. In questa rappresentazione, il generico numero negativo N viene scritto nella forma $b^n - |N|$, con $N < b^n/2$.
Per aiutarti a fissare le idee sulla rappresentazione in complemento alla base, consideriamo, ad esempio, tutti i numeri che sono rappresentabili con quattro cifre nel sistema decimale. Sappiamo che se facciamo uso di quattro cifre, possiamo rappresentare tutti i numeri naturali compresi nell'intervallo $(0, 10^4 - 1)$, ossia tutti i numeri naturali compresi fra 0 e 9999. Con riferimento a quanto detto in precedenza, i numeri positivi sono quelli compresi nell'intervallo (0, 4999) e



i negativi quelli nell'intervallo (5000, 9999). Osserviamo che 9999 corrisponde al numero −1, 9998 al numero −2 e così via, fino a 5000 che corrisponde al numero −5000. Dal momento che lo zero è stato incluso nei numeri positivi, il numero +5000 non è rappresentabile usando solo quattro cifre.
La rappresentazione in complemento alla base, apparentemente superflua, presenta un importante vantaggio pratico. In questa rappresentazione, infatti, le operazioni di addizione e sottrazione tra i numeri relativi si possono ridurre tutte ad addizioni, con l'avvertenza che, nella somma di due numeri di n cifre, va ignorata un'eventuale cifra di posto $n+1$.

Esempio 15. Per effettuare la sottrazione:

$$3475 - 2747$$

si trasforma 2747 nel suo complemento alla base 10:

$$(10000 - 2747) = 7253$$

e poi si effettua la somma tra il sottraendo e il complemento del sottrattore:

$$3475 + 7253 = (1)0728 = 728$$

Esempio 16. Per effettuare la sottrazione:

$$2747 - 3475$$

si trasforma 3475 nel suo complemento a 10, e sommando, si ottiene:

$$2747 + (10000 - 3475) = 2747 + 6525 = 9272.$$

Poiché il risultato è maggiore di 4999, esso è negativo; il suo valore in modulo e segno è dato da $-(10000 - 9272) = -728$.

Il complemento a due di un numero binario di n bit si ottiene invertendo ogni bit e aggiungendo 1 al risultato. Ad esempio, il complemento a due di 1011011 è $0100100 + 1 = 0100101$.
Osservazione: le regole per l'addizione binaria verranno date nella sezione 2; prendiamo per buono, al momento, il fatto che $0+1$ sia uguale a 1.
Osservazione importante: il bit di segno, sia nei numeri negativi che in quelli positivi, può essere ripetuto un numero qualsiasi di volte senza variare il valore del numero rappresentato.
Per i numeri positivi, infatti, in cui il bit di segno è 0, la ripetizione del bit 0 equivale all'aggiunta di zeri non significativi (01, cioè +1, equivale a 001, 0001...). Per i numeri negativi, che provengono dalla complementazione di numeri positivi, gli zeri non significativi si trasformano in altrettanti 1; così 101, cioè −3 (che proviene dalla complementazione di 011) equivale a 1101, 11101...(che provengono da 0011, 0011...).

Esempio 17. Il numero $(+43)_{10}$ si rappresenta in complemento a due con 0101011, essendo il primo bit quello di segno.

Il numero $(-43)_{10}$ si rappresenta con 1010101, essendo il primo bit quello di segno (1 perché il numero è negativo). Per risalire dalla rappresentazione in complemento a due del numero negativo 1010101 a quella del suo opposto (positivo), basta effettuare il complemento a due.
Per quanto detto nell'osservazione precedente, 0101011 si può scrivere come 00101011 oppure 000101011 ecc... Allo stesso modo, 1010101 si può scrivere come 11010101 oppure 111010101 ecc...

Domanda 7. Quali numeri decimali rappresenta la stringa 111111 nella rappresentazione in modulo e segno e in quella in complemento a due?

1.8. Rappresentazione dei numeri in virgola mobile

I calcolatori rappresentano i numeri binari con un limitato numero di bit, dipendente dalla capacità di alcuni circuiti detti "registri", che esamineremo in dettaglio nell'unità 4.
In un calcolatore con registri a 32 bit, sono rappresentabili i numeri interi con segno da -2^{32} a $2^{31} - 1$. Se vogliamo invece rappresentare i numeri con la parte intera e quella frazionaria, dobbiamo dividere la stringa di 32 bit in due sottostringhe – una per la parte intera, e una per la parte frazionaria – riducendo quindi sensibilmente l'insieme dei numeri rappresentabili.
Nei calcoli tecnico–scientifici, nei quali è necessario poter rappresentare numeri razionali sia molto piccoli che molto grandi, la rappresentazione dei numeri mediante una stringa di bit con virgola in posizione fissa non è assolutamente sufficiente. Si utilizza allora una rappresentazione dei numeri chiamata "rappresentazione in virgola mobile", consistente nella individuazione di ogni numero N mediante una coppia di numeri: l'"esponente" (e) e la "mantissa" (m), tali che il valore $V(N)$ del numero sia dato da:

$$V(N) = b^e * m$$

essendo b la base del sistema di numerazione usato. Di norma, si usa poi una rappresentazione "normalizzata" della mantissa, in modo che tutte le sue cifre siano significative:

$$b^{-1} \leq m < 1$$

che per il sistema decimale equivale a $0,1 \leq m < 1$ e per il sistema binario a $0,5 \leq m < 1$.

Esempio 18. Il numero decimale 123,456 deve essere prima trasformato in $0,123456 * 10^3$ allo scopo di normalizzare la mantissa, e quindi rappresentato dalla coppia <esponente; mantissa >:< 3; 0,123456 >.

La stessa tecnica si applica ai numeri binari. Notiamo che la rappresentazione esponente–mantissa comporta un segno per l'esponente e uno per la mantissa. La

convenzione più semplice è quella di rappresentare l'esponente in complemento a due, e la mantissa in modulo e segno, secondo il formato seguente:

$$< S >< \text{esponente} >< \text{mantissa} >$$

essendo S il segno della mantissa. Usando il formato normalizzato della mantissa, la sua parte intera che è sempre 0, non viene rappresentata, per cui il numero è identificato da:

$$N = (-1)^s * 0, m * 2^e.$$

Nei calcolatori con registri a 32 bit, di norma si assegna un bit ad S, 8 bit all'esponente e 23 bit alla mantissa. In questo modo si possono rappresentare i numeri positivi compresi tra $0,1 * 2^{-128}$ e $0,1\ldots1 * 2^{127}$, e i numeri negativi compresi tra $-0,1\ldots1 * 2^{127}$ e $-0,1 * 2^{-128}$. I numeri in valore assoluto minori di $0,1 * 2^{-128}$ e maggiori di $0,1\ldots1 * 2^{127}$ non sono rappresentabili; nel primo caso si parla di *underflow* ed il numero viene approssimato a 0, nel secondo caso si ha *overflow* e quindi errore.
La risoluzione, intesa come minima variazione relativa tra due numeri consecutivi, è determinata dal numero delle cifre della mantissa; pertanto, l'errore che si commette nel rappresentare un numero reale è, in valore relativo, dato da:

$$e < 2^{-r}$$

essendo r le cifre della mantissa.
Osservazione importante: riportando i valori sopra esposti in decimale e tenendo conto che n cifre binarie corrispondono a circa $0,3 * n$ cifre decimali, possiamo concludere che la rappresentazione in virgola mobile a 32 bit permette di rappresentare numeri tra 10^{-38} e 10^{+38}, con un errore minore di 10^{-7}.

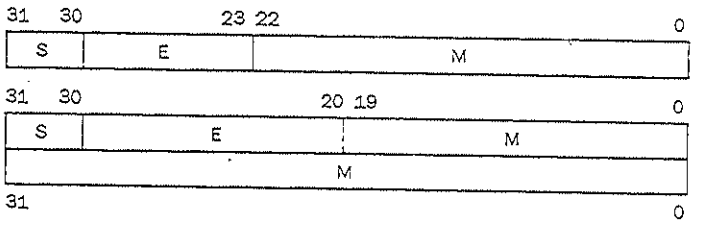
Esempio 19. Il numero $+10111,0011$ viene trasformato prima in $+0,101110011 * 2^5$ (vedi l'analogo esempio 18) e rappresentato con:

$$< 0 >< 00000101 >< 10111001100000000000000 > .$$

Domanda 8. Rappresenta in virgola mobile il numero binario $-0,000101110011$.

1.9. La normativa IEEE

Da quanto detto precedentemente si deduce che sono possibili diverse rappresentazioni in virgola mobile, sia per quanto riguarda il numero di bit complessivi, sia per come essi sono ripartiti tra mantissa ed esponente, sia per come si può rappresentare l'esponente. Per facilitare la trasportabilità del software da una macchina all'altra l'Institute of Electrical and Electronic Engineering (IEEE) ha proposto uno standard a cui attenersi per la rappresentazione dei numeri in virgola mobile. Questo standard prevede un formato in singola precisione, uno in doppia precisione di tipo semplice o esteso. In questa sede descriveremo solo i due standard semplici che prevedono il seguente formato:



Nel caso di singola precisione si hanno 32 bit di cui 8 per l'esponente e 23 per la mantissa, mentre per la doppia precisione si hanno 64 bit di cui 11 per l'esponente e 52 per la mantissa. L'esponente non è espresso in complemento a due ma in forma cosiddetta polarizzata usando una costante di polarizzazione P pari a $2^{nE-1} - 1$, dove nE rappresenta il numero di bit dell'esponente. Nel caso di precisione singola è $P = 2^7 - 1 = 127$, per cui l'esponente può assumere valori compresi tra -127 (rappresentato, nella forma polarizzata, da 00000000) e $+127$ (rappresentata da 11111111). La mantissa è normalizzata nell'intervallo $1 \leq M < 2$; la sua parte intera è, pertanto, sempre costituita da un bit uguale ad 1, la cui rappresentazione è omessa (hidden bit): in questo modo, pur usando 23 bit per rappresentare la mantissa, questa risulta costituita da 24 bit significativi. Secondo questa convenzione il numero N è rappresentato dall'espressione:

$$N = (-1)^s * 1, M * 2^{E-P}.$$

Osservazione: abbiamo detto, all'inizio di questo paragrafo, che le cifre disponibili per rappresentare i numeri sono legate alle dimensioni dei registri. La rappresentazione in doppia precisione è tuttavia possibile perché vengono usati due registri del calcolatore come se fossero un unico registro a 64 bit.

Esempio 20. Il numero decimale 1023 è rappresentato come:

0	10001010	11111111100000000000000

Infatti si può scrivere: $1023_{10} = 1111111111 = 1,1111111111 * 2^9$, per cui:

$$M = 11111111100000000000000 \quad \text{e} \quad E = 127 + 9 = 138_{10} = 10001010.$$

Domanda 9. Cosa bisogna aggiungere alla rappresentazione del numero +10111,0011 fornita nell'esempio 19, per trasformarla in una rappresentazione in virgola mobile in doppia precisione?

Una particolarità della normativa IEEE è quella di trattare in modo esplicito valori numerici particolari; per far ciò i due valori estremi dell'intervallo di definizione dell'esponente (0 e $2^{nE} - 1$) non sono utilizzati per la rappresentazione di numeri normali ma per rappresentare situazioni anormali secondo la seguente tavola:

Segno	Esponente	Mantissa	Significato
0	0	0	$+0$
1	0	0	-0
0	Valore max	0	$+\infty$
1	Valore max	0	$-\infty$
0/1	Valore max	$\neq 0$	non è un numero
0/1	0	$\neq 0$	numero non normalizzato

Abbiamo analizzato, in questa sezione, numerosi argomenti; vogliamo ora mettere in evidenza quelli più importanti ai fini del nostro corso.
Fondamentale è la conoscenza del sistema binario, e con esso della rappresentazione dei numeri con segno e dei numeri in virgola mobile. I sistemi ottali ed esadecimali sono importanti in quanto, di norma, stringhe di bit vengono "condensate" trasformandole in uno di questi due sistemi. Tra le regole di conversione, che permettono di passare da un sistema di numerazione all'altro, sono peculiari quelle che riguardano le conversioni con il sistema decimale che è la nostra "lingua madre" numerica.

2. OPERAZIONI ARITMETICHE

2.1. Introduzione

Per la comprensione del funzionamento dei circuiti che realizzano le operazioni aritmetiche all'interno del calcolatore, è necessario conoscere le regole per effettuare le operazioni stesse direttamente sui numeri binari.
Le operazioni aritmetiche sui numeri binari seguono le stesse regole di quelle sui numeri decimali, ma la loro attuazione è più semplice, considerando che le cifre in gioco sono sempre e soltanto due. Occorre, di contro, prestare attenzione al numero di bit di cui sono costituiti gli operandi e il risultato di ogni operazione, tenendo presente che le operazioni sono eseguite da una macchina, in circuiti che hanno una capacità fissa di cifre.
Esaminiamo di seguito le quattro operazioni aritmetiche fondamentali

2.2. Addizione

Le regole per l'addizione di due bit sono:

$$0+0=0$$
$$0+1=1$$
$$1+0=1$$
$$1+1=0 \quad \text{con riporto (carry) di 1.}$$

L'ultima regola si può anche scrivere $1+1=10$.
L'addizione di due numeri binari positivi si esegue come nel sistema decimale, tenendo conto dei riporti.

Esempio 21. La somma dei numeri 00101010 e 01011011 è data da:

1 1 1 1 0 1 0	riporto
0 0 1 0 1 0 1 0	addendo A
0 1 0 1 1 0 1 1	addendo B
1 0 0 0 0 1 0 1	somma A+B.

Quando un calcolatore esegue una addizione di questo tipo si può verificare una situazione di trabocco (overflow) se la somma è costituita da un numero di bit maggiore di quello consentito, ad esempio, dal registro destinato a contenere la somma stessa.
Tale situazione si verifica quando c'è un riporto nella coppia di cifre più significative, come mostra l'esempio seguente.

Il suo quadrato viene calcolato nel seguente modo:

$$< S > = 0 (< S1 > = < S2 >);$$
$$< \text{esponente} > = 00000011 + 00000011 = 00000110$$
$$< \text{mantissa} > = 0010101 * 0010101 = 110111001.$$

Il risultato è pertanto:

0	00000110	00000000000000110111001
$< S >$	$< \text{esponente} >$	$< \text{mantissa.} >$

In forma normalizzata il risultato si scrive:

0 00010100 11011100100000000000000.

Le operazioni di somma e sottrazione avvengono tra le mantisse e lasciano invariati gli esponenti, se questi sono uguali. Nel caso in cui gli esponenti sono diversi, le operazioni tra le mantisse vanno precedute da uno scorrimento verso destra della mantissa relativa all'esponente minore, del numero di posizioni necessario per rendere uguali i due esponenti. Si ha dunque:

$< S >$ = valore risultante dalla somma, o differenza, delle mantisse;
$< \text{esponente} > = < \text{esponente1} > = < \text{esponente2} >;$
$< \text{mantissa} > = < \text{mantissa1} > + (-) < \text{mantissa2} >.$

Nel caso di rappresentazione normalizzata, la somma delle due mantisse può traboccare di un bit ($0,5 \leq M_1 + M_2 < 2$); in tal caso il risultato va normalizzato incrementando di 1 l'esponente e scalando a destra la mantissa.

Esempio 30. La somma tra:

$$N1 = 0; 00000011; 10111010000000000000000$$

ed:

$$N2 = 0; 00000101; 11100010000000000000000$$

viene effettuata confrontando prima gli esponenti. Essendo:

$$< \text{esponente2} > = < \text{esponente1} > + 2,$$

facciamo scorrere $< \text{mantissa1} >$ di due posti verso destra ponendo, nello stesso tempo, $< \text{esponente1} > = < \text{esponente2} >$. Si ha quindi:

$$N1 = 0; 00000101; 00101110100000000000000$$

Si ha a questo punto:

$< S > = 0;$
$< \text{esponente} > = 00000101;$
$< \text{mantissa} > = 0010111010 + \cdots + 1110001000\ldots = (1)0001000010\ldots$

Il risultato è pertanto:

0	00000110	10001000010000000000000
$< S >$	$< \text{esponente} >$	$< \text{mantissa} >$

Domanda 14. Qual è il risultato di $N1 * N2$, essendo $N1$ ed $N2$ quelli dell'esempio 30?

Ed ora, dopo tante operazioni, prova a risolvere un semplice esercizio di geometria.

Esercizio 4. Trovare il volume di una piramide a base quadrata di lato $l = 10010m$ ed altezza $h = 100 * l - 1000m$. Tutti i dati sono in binario; i calcoli vanno effettuati in binario, con carta e matita, senza preoccuparsi della lunghezza dei risultati intermedi e finali (che, ovviamente, saranno tutti in binario).

In questa sezione abbiamo imparato ad effettuare le operazioni aritmetiche fondamentali direttamente in binario, e nei diversi sistemi di numerazione. Scopo della sezione non è tanto di invogliarti all'uso del sistema binario nella vita di tutti i giorni, quanto di farti comprendere come procedono i calcolatori elettronici per effettuare le operazioni usando tali sistemi.

3. I CODICI

3.1. Introduzione

I calcolatori elaborano informazioni rappresentate, o codificate, in forma binaria. Nelle prime due sezioni abbiamo visto come vengono codificati i numeri e come, su queste rappresentazioni codificate, vengano effettuate le operazioni aritmetiche.

Numeri e operazioni aritmetiche sono però soltanto un tipo di informazione ed un tipo di elaborazione tra tutte quelle che i calcolatori possono trattare ed compiere. Altre informazioni coinvolgono la rappresentazione delle lettere dell'alfabeto e di alcuni caratteri speciali, su cui vengono compiute operazioni di ordinamento, confronto, traduzione, ecc.

Nel linguaggio normale, la parola "codice" evoca spesso qualcosa dal significato oscuro, comprensibile solo agli iniziati. I militari usano scrivere i loro messaggi in codice; i politici parlano spesso in codice; i medici e gli avvocati (ma anche i tecnici) si esprimono secondo particolari codici. "Codice" indica quindi un particolare modo di comunicare, secondo simboli e regole precise, che debbono essere conosciute per comprendere il senso delle comunicazioni.

Una diversa accezione del termine, più vicina a quella che ci interessa per i nostri scopi, indica una stringa di lettere e/o cifre per individuare una persona o una cosa. Sono "codici" di questo tipo: i codici fiscali, i numeri telefonici, le targhe automobilistiche, i CAP...

In generale, codice è un insieme C di simboli e di regole adottato per rappresentare gli elementi di un insieme di simboli o oggetti C'. Sono pertanto codici, ad esempio, anche le parole di un linguaggio, i numeri decimali, i numeri binari; simboli di questi codici sono, rispettivamente, le lettere dell'alfabeto, le cifre decimali, i bit.

Parole di un codice sono le combinazioni di simboli che hanno un significato. Così, ad esempio, "casa" è una parola della lingua italiana, mentre "saca" non lo è, in quanto questa combinazione di simboli non ha senso nella nostra lingua.

Codifica è l'operazione per cui una parola di C viene associata a un elemento di C'.

Decodifica è l'operazione per cui viene individuato l'elemento di C' corrispondente ad una parola di C.

Un codice C si dice non ambiguo quando la corrispondenza tra le parole di C e gli elementi di C' è univoca, nel senso che ogni elemento che appartiene a C' corrisponde ad una e solo una parola di C.

Sia N il numero degli elementi di C', b il numero di simboli del codice C, ed n la lunghezza (costante) delle parole di C. Il minimo valore (m) di n che rende il codice C non ambiguo eguaglia il numero di cifre necessarie per scrivere N nel sistema di numerazione a base b. In altre parole, deve essere $b^m \geq N$; m è cioè il più piccolo intero non minore del $\log_b N$.

Ad esempio, per codificare 500 elementi in un codice binario occorrono parole di 9 bit, essendo $9 = \lceil \log_2 500 \rceil$, ($2^9 = 512 > 500 > 2^8 = 256$).

Con i simboli introdotti, un codice si dice irridondante se $n = m$; ridondante se $n > m$, ambiguo se $n < m$.

Ad esempio, se codifichiamo 16 elementi di un insieme con 4 bit, otteniamo un codice irridondante; se codifichiamo i 16 elementi con 5 bit, otteniamo un codice ridondante; se infine codifichiamo i 16 elementi con 3 bit, otteniamo un codice ambiguo (perché potremo disporre solo di 8 combinazioni, e dovremo associare due dei 16 elementi a ciascuna di esse). Di seguito esamineremo alcuni dei più importanti codici binari, con lunghezza fissa della parola, non ambigui. I codici ambigui non hanno alcuna importanza nella nostra disciplina.

3.2. Codici binari ridondanti e irridondanti

Un codice binario C permette la codifica degli elementi di un generico insieme C' mediante una stringa di bit di lunghezza n tale che 2^n sia non minore del numero N degli elementi di C'.

Secondo quanto visto nel paragrafo precedente, detto n il numero di bit di una parola, e posto $m = \log_2 N$, il codice binario è irridondante se $n = m$, e ridondante se $n > m$: in quest'ultimo caso, i $k = m - n$ bit in più sono detti bit di controllo del codice.

Si definisce distanza di Hamming h di un codice il minimo numero di bit di cui differiscono due parole qualsiasi del codice; per i codici irridondanti, $h = 1$; per quelli ridondanti, $h > 1$.

I codici ridondanti hanno grande importanza pratica, in quanto la ridondanza permette di riconoscere, o addirittura di correggere, gli errori (un errore è l'inversione di un bit che si manifesta, di norma, durante la trasmissione delle informazioni sotto forma di segnali elettrici; gli errori possono essere diminuiti, ma non eliminati completamente). In particolare, i codici con $h > 1$ sono in grado di rivelare gli errori su $R = h - 1$ bit.

Esempio 31. Consideriamo un generico codice binario, e indichiamo con O una parola di codice, e con X una configurazione di bit che non è una parola del codice (cioè una configurazione errata). Ricordando la definizione di codice di Hamming, se $h = 3$, due parole di codice sono separate da due configurazioni errate:

$$O_1 \quad X \quad X \quad O_2.$$

Queste configurazioni possono derivare da un bit errato nella parola più vicina, o da due bit errati in quella più lontana. Così, dalle parole a distanza tre 00000 e 01011 si può originare la configurazione 01000 per un errore sul secondo bit di 00000, o per due errori sugli ultimi due bit di 01011.

I codici con $h \geq 3$ possono anche essere impiegati per correggere gli errori: la correzione avviene interpretando la configurazione errata come proveniente dalla parola più vicina.

Esempio 32. Volendo usare il codice dell'esempio 31 come correttore d'errore, faremo derivare le configurazioni errate dalle parole più vicine, nel modo illustrato:

$$\underline{O_1 \quad X} \qquad\qquad \underline{X \quad O_2}.$$

Pertanto, se 00000 e 01011 sono due parole di codice, la configurazione 10000 verrà riconosciuta come 00000, e la configurazione 00011 verrà riconosciuta come 01011.

Se usiamo un codice con $h = 4$, possiamo correggere gli errori dovuti all'inversione di un bit, e riconoscere quelli dovuti all'inversione di due bit (che non possono essere corretti in quanto possono derivare da un doppio errore intervenuto e sulla parola che si trova a destra della configurazione errata, o su quella a sinistra):

$$\underline{O_1 \quad X} \quad X \quad \underline{X \quad O_2}.$$

Pertanto, se 0000000 e 0001111 sono due parole di codice, la configurazione 0000001 verrà riconosciuta come 0000000, ma la configurazione 0000011 non potrà essere corretta, in quanto potrebbe derivare dall'inversione degli ultimi due bit della parola 0000000 oppure dall'inversione del quarto e quinto bit della parola 0001111.
In generale, chiamando h la distanza del codice, R la capacità di rivelazione di errori e C la capacità di correzione di un codice, valgono le relazioni $2C + 1 \leq h$ e $C + R = h - 1$ come è immediato ricavare dalla figura seguente in cui O_1 e O_2 sono due parole di codice a distanza minima h e i cerchi rappresentano i punti del piano in cui sono corrette o rivelate parole non del codice.

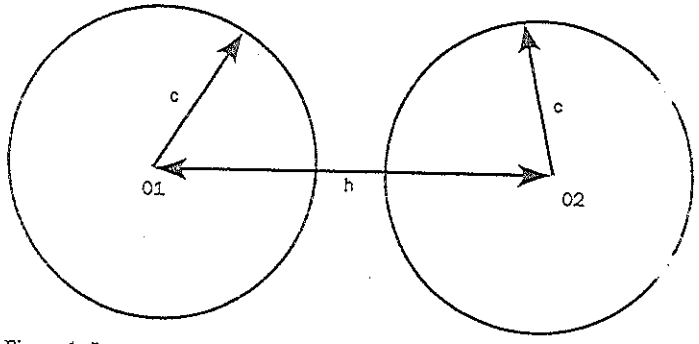


Figura 1. *Rappresentazione grafica di due parole a distanza h.*

Domanda 15. Un codice binario ridondante con distanza di Hamming $h = 6$ può essere utilizzato come rivelatore o come correttore/rivelatore d'errore. Quanti bit errati può rivelare o correggere nei due casi?

I codici a rivelazione e/o a correzione d'errore sono un argomento molto complesso e importante. Qui ne abbiamo dato un rapido cenno, sufficiente a inquadrare quei codici che studieremo nei prossimi paragrafi.

3.3. Codici BCD, di Gray, ASCII

Il codice BCD (Binary Coded Decimal) è un codice irridondante per codificare le dieci cifre decimali con quattro bit, secondo il sistema di numerazione bina-

rio, è rappresentato nella tabella 1.

Cifra	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Tabella 1. *Codice BCD.*

Per codificare in BCD un numero decimale occorrono più cifre che nel sistema di numerazione binario; in compenso, le espressioni ottenute sono immediatamente interpretabili.

Esempio 33. Il numero 908 si rappresenta con la stringa:

1001 0000 1000.

Domanda 16. Come viene codificato in BCD, il numero binario 11111?

Il codice di Gray rappresenta i numeri binari di n bit in modo che due numeri successivi vengono codificati con stringhe che differiscono per un solo bit. Ad esempio, il codice di Gray a due bit è: 00, 01, 11, 10.
Per passare dal codice di Gray a n bit a quello a $n+1$ si procede così:

- si riscrive il codice a n bit in forma speculare, ovvero dall'ultima configurazione alla prima;
- si premette uno 0 davanti alle vecchie configurazioni;
- si premette un 1 davanti alle nuove configurazioni (quelle cioè scritte in forma speculare).

Per questa ragione il codice di Gray si chiama anche "codice riflesso".

Esempio 34. La costruzione del codice di Gray a tre bit, partendo da quello a due bit, è molto semplice:

- codice a due bit: 00 01 11 10;
- codice speculare: 10 11 01 00;
- codice a tre bit: 000 001 011 010 110 111 101 100.

Nella tabella 2 è mostrato il codice di Gray a quattro bit, per rappresentare i numeri decimali da 1 a 15.
Esistono appositi algoritmi, che possono essere implementati da circuiti estremamente semplici, per passare dal codice binario al codice di Gray.

Decimale	Gray
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

Tabella 2. *Codice di Gray a quattro bit.*

Il codice ASCII (American Standard Code for Information Interchange) è un codice binario a sette bit; è quindi in grado di rappresentare $2^7 = 128$ simboli diversi: le lettere maiuscole e minuscole, nonché un gran numero di segni speciali e caratteri di controllo.
Il codice ASCII, riportato nella tabella 3, è il codice alfanumerico più usato, sia per la rappresentazione dei dati all'interno del calcolatore che per lo scambio di informazioni tra sistemi diversi. Nella tabella, ad ogni simbolo corrisponde una stringa di sette bit: i primi tre sono quelli sulla colonna dove si trova il simbolo; gli ultimi quattro sono quelli della relativa riga. Ad esempio, il numero 7 è codificato con 0110111.

Domanda 17. Qual è la codifica ASCII di "ASCII", espressa in esadecimale?

In molti casi, ai sette bit del codice ASCII viene aggiunto un ottavo bit, che può avere due significati diversi. Se è usato per la trasmissione di informazioni, l'ottavo bit è un bit ridondante, che comporta $h = 2$, e permette di rivelare un errore su un singolo bit di un carattere.
In altri casi, l'ottavo bit è invece usato per aumentare il numero dei simboli codificabili, portandoli a 256 (in questo caso, i simboli standard hanno l'ottavo bit uguale a 0, e i simboli aggiuntivi hanno l'ottavo bit uguale a 1).

Righe	Colonne	0	1	2	3	4	5	6	7
b7b6b5		000	001	001	011	100	101	110	111
	b4b3b2b1								
0	0000	NUL	DLE	SP		0	P	`	p
1	0001	SHO	DC1	—	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	MEM	)	9	I	Y	i	y
10	1010	LF	SUB	*	:	J	Z	j	z
11	1011	VT	ESC	+	;	K	—	k	{
12	1100	FF	FS	,	>	L	\	l	|
13	1101	CR	GS	-	=	M	—	m	}
14	1110	SO	RS	.	>	N	^	n	~
15	1111	SI	US	/	?	O	"	o	DEL

Tabella 3. *Codice ASCII.*

3.4. Codici ridondanti e codici di Hamming

Come già detto, i codici ridondanti sono quelli con $h > 1$, e possono rivelare e/o correggere uno o più bit errati.
Il più semplice codice ridondante è il codice di parità, che ha $h = 2$, e si ottiene premettendo un bit di parità p ad un codice irridondante. Il valore di p deve essere tale da rendere il numero complessivo di bit 1 (peso) di ogni parola sempre pari (codici a parità pari) o sempre dispari (codici a parità dispari).

Esempio 35. Il codice BCD della tabella 1 può essere trasformato in un codice a parità pari premettendo ad ogni sua parola il bit di parità. Ad esempio, le configurazioni 0111 e 0110 diventano, così, 10111 e 00110 rispettivamente. Gli errori rivelati sono quelli che intervengono sulle configurazioni con un numero dispari di bit, perché queste ultime non conservano la parità.

Esercizio 4. Scrivere il numero $(327)_{10}$ in ASCII, con controllo di parità dispari. Esprimere la stringa in binario e in esadecimale.

Una "famiglia" di codici ridondanti importante per la sua capacità di correggere certi errori di trasmissione è costituita dai "codici di Hamming". L'argomento, anche se trattato per sommi capi, è piuttosto complesso, e non è comunque essenziale per il nostro corso. Puoi pertanto tralasciarlo, o leggerlo per sola conoscenza, riprendendo lo studio dalla fine dell'esercizio 5.
I codici di Hamming sono quelli con $h > 2$, cioè con capacità di autocorrezione di errori. Il più semplice codice di Hamming ha $h = 3$, ed è strutturato su parole di $m = n + k$ bit, dove n è il numero di bit di un codice irridondante, e k il numero dei bit di controllo necessari per trasformare il codice irridondante nel codice con $h = 3$.
Poiché con k bit si realizzano 2^k combinazioni diverse, per poter rivelare la presenza di errori sugli $n+k$ bit della nuova parola, dovrà essere $2^k \geq n+k+1$. In altre parole, il numero di bit n controllabili mediante k bit di controllo soddisfa la condizione:

$$n \leq 2^k - k - 1.$$

In pratica, dette $a_1, a_2, a_3, \ldots, a_m$ gli m bit che costituiscono ogni parola del codice di Hamming, i k bit di controllo sono allocati nelle posizioni di indice 2^i ($i = 0, 1, \ldots, k-1$). Il controllo esercitato con questi bit si effettua nel modo seguente:

- si esprimono in binario gli indici che indicano le posizioni di ognuno degli m bit;
- si considerano tutti i bit nelle posizioni il cui indice contiene un 1 di peso 2^i;
- il bit in posizione 2^i controlla la parità della combinazione dei bit precedentemente considerati.

Chiariamo questa regola astrusa con un esempio.

Esempio 36. Trasformare il codice ASCII a 7 bit nel codice di Hamming con $h = 3$; codificare quindi nel codice ottenuto la cifra decimale 0.
Per essere $n = 7$, risulta $k = 4$ ed $m = 11$. La struttura di una parola di codice è costituita da 11 bit:

$$a_1\, a_2\, a_3\, a_4\, a_5\, a_6\, a_7\, a_8\, a_9\, a_{10}\, a_{11}.$$

I quattro bit di parità sono nelle posizioni 2^i ($i = 0, 1, 2, 3$), cioè nelle posizioni a_1, a_2, a_4, a_8. Indicando con c i bit di controllo, ed esprimendo tutti gli indici in binario, la struttura della parola è:

$$c_{0001}\, c_{0010}\, a_{0011}\, c_{0100}\, a_{0101}\, a_{0110}\, a_{0111}\, c_{1000}\, a_{1001}\, a_{1010}\, a_{1011}.$$

Il bit di controllo nella posizione 2^i controlla la parità di tutti quelli che, nel proprio indice, contengono 1 nella posizione i. Quindi:

- c_1 controlla la parità di $c_1, a_3, a_5, a_7, a_9, a_{11}$;
- c_2 controlla la parità di $c_2, a_3, a_6, a_7, a_{10}, a_{11}$;
- c_4 controlla la parità di c_4, a_5, a_6, a_7;
- c_8 controlla la parità di c_8, a_9, a_{10}, a_{11}.

La cifra 0, in ASCII, è 0110000; applicando le relazioni precedenti, si ha che c_1 controlla la parità della combinazione $c_1 a_3 a_5 a_7 a_9 a_{11}$, cioè $c_1 01000$, da cui $c_1 = 1$. Analogamente si ottiene $c_2 = 1, c_4 = c_8 = 0$.
La parola di codice è pertanto: 11001100000.
In ricezione, viene costruito un "numero di controllo" N_c di k bit, dove il bit j ($j = 0, 1, \ldots, k-1$) è 0 se è verificata la parità dei bit nelle posizioni il cui indice contiene 2^j. Se $N_c = 0$, la parola di codice ricevuta è corretta (perché sono state rispettate tutte le parità). Se invece $N_c = E \neq 0$, si dimostra che il bit in posizione E è errato, e pertanto va invertito.
I bit del numero di controllo N_c vengono calcolati mediante un'operazione chiamata "somma modulo 2" che agisce su un numero qualsiasi di bit e dà come risultato:

- 1, se i bit 1 sono in numero dispari;
- 0, se i bit 1 sono in numero pari.

Ad esempio, somma modulo $2\,(1,0,1,1,0) = 1$; somma modulo $2\,(1,1) = 0$.
Vedremo più in dettaglio questa operazione nella prossima Unità.
Sempre più difficile? Ricorriamo ancora ad un esempio, riprendendo il codice dell'esempio 36.

Esempio 37. Il numero di controllo N_c è formato da quattro bit, ognuno dei quali è la somma modulo 2 dei bit controllati.

N_c1 =somma modulo $2\,(c_1, a_3, a_5, a_7, a_9, a_{11})$;
N_c2 =somma modulo $2\,(c_2, a_3, a_6, a_7, a_{10}, a_{11})$;
N_c3 =somma modulo $2\,(c_4, a_5, a_6, a_7)$;
N_c4 =somma modulo $2\,(c_8, a_9, a_{10}, a_{11})$.

Supponiamo che in ricezione abbiamo ricevuto la parola 11001100000. Formiamo il numero di controllo:

N_c1 =somma modulo $2\,(1,0,1,0,0,0) = 0$;
N_c2 =somma modulo $2\,(1,0,1,0,0,0) = 0$;
N_c3 =somma modulo $2\,(0,1,1,0) = 0$;
N_c4 =somma modulo $2\,(0,0,0,0) = 0$.

Pertanto $N_c = 0000$; la parola è esatta. Togliendo i bit di controllo, abbiamo 0110000.
Se invece avessimo ricevuto la parola 11001100100 il numero di controllo diventerebbe:

N_c1 =somma modulo $2\,(1,0,1,0,1,0) = 1$;
N_c2 =somma modulo $2\,(1,0,1,0,0,0) = 0$;
N_c3 =somma modulo $2\,(0,1,1,0) = 0$;
N_c4 =somma modulo $2\,(0,1,0,0) = 1$.

Pertanto, $N_c = 1001$ e il bit errato è quello in posizione 9. Invertendolo, si ha 11001100000 (che è la parola esatta!).

Sui codici ridondanti, autocorrettori e rivelatori di errore, abbiamo fatto un rapido accenno, molto semplificato; sull'argomento esistono trattazioni approfondite che richiedono però una buona padronanza dell'algebra moderna. Per chiudere, ti

proponiamo di rispondere a una domanda e di provare a risolvere un esercizio di media difficoltà. Cominciamo con la domanda.

Domanda 18. Si può usare un codice di Hamming esclusivamente come rivelatore di errore?

Esercizio 5. Trasformare il codice BCD in un codice di Hamming con $h = 3$. (suggerimento: trova il numero k e scrivi il formato della parola...).

In questa sezione ci siamo occupati di codici, cioè un modo per rappresentare in binario simboli qualsiasi, cifre comprese. Nelle prime due sezioni, i simboli 0 e 1 erano cifre di un sistema di numerazione, cioè avevano un significato numerico, e le stringhe formate con essi potevano essere adoperate nelle operazioni aritmetiche. In questa sezione, 0 e 1 non sono più cifre, ma solo simboli: le stringhe di 0 e 1 cioè non sono numeri, anche quando codificano cifre, e non possono essere adoperate direttamente nei calcoli (occorre prima tradurle nei numeri binari corrispondenti).
Non tutti gli argomenti trattati in questa sezione hanno la stessa importanza. Sono fondamentali i concetti di codici e di codifica, i codici BCD ed ASCII; sono da ritenersi complementari invece gli argomenti relativi ai codici di Gray e di Hamming.
Per terminare questa unità, ti proponiamo ora di risolvere quest'esercizio, non difficile ma utile per la comprensione del modo in cui vengono rappresentati e trasformati i dati.

Esercizio 6. Sulla tastiera di un terminale vengono battuti i caratteri:

18+6=

I caratteri, trasformati prima in ASCII, vengono successivamente convertiti in binario nel calcolatore, che esegue il calcolo indicato e fornisce il risultato. Quest'ultimo, trasformato in ASCII, viene poi inviato a un terminale che visualizza il numero 24.
Indicare:

1. la codifica in ASCII di "18+6=";
2. la trasformazione nel sistema binario dei numeri 18 e 6;
3. il risultato dell'addizione in binario;
4. la codifica del risultato in ASCII.

4. CONCLUSIONI

4.1. Sommario

In questa prima unità abbiamo imparato alcune nozioni teoriche fondamentali per la comprensione del corso "Architettura dei Calcolatori".
Abbiamo dapprima messo in rilievo che i circuiti del calcolatore possono riconoscere, immagazzinare ed elaborare grandezze fisiche in grado di assumere due soli valori distinti e significativi. Per questa ragione, è indispensabile che tutte le informazioni trattate dal calcolatore siano rappresentate come un insieme di valori binari.
Abbiamo quindi introdotto un simbolismo per rappresentare gli stati binari, fondato sull'uso dei simboli 0 e 1. A tali simboli, e alle loro combinazioni, abbiamo poi assegnato significati generici – come nel caso dei codici – o precisi significati numerici.
Più dettagliatamente, in questa Unità abbiamo introdotto il sistema di numerazione binario, che fa uso dei due simboli 1 e 0, ricavandone le proprietà generali a partire da quelle del sistema di numerazione decimale, a noi così familiare. Abbiamo visto come questi due sistemi abbiano in comune il fatto di essere entrambi sistemi di numerazione posizionali, e abbiamo imparato a costruire sistemi di numerazione in base qualsiasi. Tra i sistemi non decimali, abbiamo quindi preso in esame quello ottale e quello esadecimale che usano rispettivamente 8 e 16 simboli diversi.
Avendo a disposizione ben quattro sistemi di numerazione, abbiamo quindi imparato le regole che ci permettono di passare dall'uno all'altro, cioè di scrivere i numeri indifferentemente nell'uno o nell'altro sistema. Tra queste regole, le più importanti sono quelle che fanno passare dal sistema binario a quello decimale, e viceversa. Abbiamo anche appreso l'uso dei sistemi ottale ed esadecimale come sistemi pratici per scrivere in modo conciso i numeri binari.
Dopo aver limitato il nostro studio ai numeri senza segno, siamo quindi passati alla rappresentazione in binario dei numeri relativi. Successivamente, legando la rappresentazione dei numeri binari alle dimensioni fisiche degli organi del calcolatore destinati a contenerli, abbiamo introdotto un metodo di rappresentazione dei numeri relativi, quello in virgola mobile, che permette di rappresentare numeri molto grandi insieme con numeri molto piccoli. Con questo argomento abbiamo chiuso la prima sezione della nostra Unità.
Sui numeri espressi in binario, il calcolatore esegue direttamente le operazioni aritmetiche. Pertanto, dovendo studiare la costituzione interna del calcolatore, abbiamo imparato ad eseguire manualmente le operazioni aritmetiche in binario. Ci siamo soffermati in modo particolare sull'operazione di addizione, introducendo una nuova rappresentazione dei numeri binari, quella in complemento a due, che ha la proprietà di semplificare radicalmente le operazioni sui numeri relativi. Particolare attenzione abbiamo riservato al verificarsi del trabocco, un evento che può falsare i risultati delle operazioni aritmetiche, e che è originato dalla limitata capacità degli organi del calcolatore.
Il calcolatore non tratta soltanto informazioni di tipo numerico, ma anche infor-

mazioni espresse mediante caratteri dell'alfabeto e segni speciali. Anche queste informazioni, come quelle numeriche, non possono essere espresse che usando i simboli 0 e 1. La corrispondenza tra i caratteri da rappresentare e determinate stringhe di bit è ottenuta con l'uso dei codici. Dopo un breve accenno alla teoria generale dei codici, abbiamo passato in rassegna alcuni dei codici più comuni ed usati nei calcolatori: particolarmente importanti, fra tutti, il codice BCD e il codice ASCII. Abbiamo infine fatto un accenno ai codici di Hamming, la cui importanza è legata al problema della protezione dagli errori che si presentano durante la trasmissione dei dati tra due punti.

4.2. Risposte alle domande

Domanda 1. Il valore della base viene espresso con la scrittura 10.

Domanda 2. Dal fatto che è intero ed ha 1 come bit meno significativo.

Domanda 3. $(70654)_8 = 7 * 8^4 + 0 * 8^3 + 6 * 8^2 + 5 * 8 + 4 = (29100)_{10}$.

Domanda 4.
$$(\text{CAFFE})_{16} = C * 16^4 + A * 16^3 + F * 16^2 + F * 16 + E =$$
$$= 12 * 16^4 + 10 * 16^3 + 15 * 16^2 + 15 * 16 + 14 =$$
$$= (831.486)_{10}.$$

Domanda 5.
$$(4805)_{10} = (1001011000101)_2$$
$$(0,0234375)_{10} = (0,0000011)_2$$

pertanto si ha 1001011000101,0000011.

Domanda 6.

$6005/8 = 750$ resto 5
$750/8 = 93$ resto 6
$93/8 = 11$ resto 5
$11/8 = 1$ resto 3
$1/8 = 0$ resto 1

$0,6005 * 8 = 4,804$
$0,804 * 8 = 6,432$
$0,432 * 8 = 3,456$
$0,456 * 8 = 3,648.$

Pertanto se si arresta la conversione alla quarta cifra dopo la virgola si ottiene il numero ottale 13565,4633.

Domanda 7. La stringa binaria 111111, nella rappresentazione in modulo e segno rappresenta il numero decimale -31, in quella in complemento a due rappresenta il numero -1.

Domanda 8. Portando il numero in forma normalizzata otteniamo:

$$-0,000101110011 = -0,1011110011 * 2^{-3}$$

e $S = 1$; -3 va rappresentato in complemento a 2 con 8 bit (11111101). Pertanto il numero diventa:

< 1 >< 11111101 >< 10111100110000000000000 > .

Domanda 9. Occorre prima riportare il numero nella forma $1,01110011 * 2^4$, aggiungere all'esponente 4 lo spiazzamento $1023(2^{nB-1} = 1024)$, ed estendere la mantissa a 52 bit ottenendo:

< 0 >< 10000000011 >< 0111001100000000000000000000000000000000000000000000 > .

Domanda 10.

11111110	-2
00011100	+28
00011010	+26.

Domanda 11. Ricordando che $A - B = A + (-B)$, complementiamo il sottrattore 11111001, ottenendo 000000111, dopo di che possiamo sommare i due numeri:

11110000	-16
00000111	-(-7)
11110111	-9.

Domanda 12. Il prodotto avrà segno negativo, essendo i bit di segno dei due operando differenti. Per quanto riguarda i moduli si ha:

1110010 *	114
0001111	15
1110010	
1110010	
1110010	
1110010	
11010101110	1710

Il risultato finale sarà quindi con 1 bit in più a sinistra per aggiungere il segno 111010101110.

Domanda 13. Poichè i bit di segno uguali, il bit di segno del quoziente è 0. Per riguarda i moduli si ha 1100101 : 10000 = 110 con resto 101 (l'operazione è immediata, perché il divisore è una potenza di 2).

Domanda 14.
$$< S > = 0$$
$$< \text{esponente} > = 11 + 101 = 1000$$
$$< \text{mantissa} > = 1011101 * 1110001 = 10100100001101.$$

Domanda 15. Il codice può essere impiegato:

- come rivelatore di errore, rivelando fino a 5 bit errati;
- come correttore/rivelatore d'errore, per correggere fino a 2 bit errati e rivelare un errore su un terzo bit errato.

Domanda 16. $(11111)_2 = (31)_{10}$; e quindi in BCD: 0011 0001.

Domanda 17. 41 53 43 49 49.

Domanda 18. Certo: in questo caso basta verificare che N_c sia diverso da zero per rivelare la presenza di errore. In questo caso vengono rivelati anche errori doppi.

4.3. Soluzioni degli esercizi

Esercizio 1. $(1011,001)_2 = (10^{11})_2 + (10^1)_2 + (10^0)_2 + (10^{-11})_2$

Esercizio 2. Dall'equazione $m = \lceil n * \log_2 10 \rceil$ si ricava:

$$n = \lceil m/\log_2 10 \rceil = \lceil 0,30 * m \rceil,$$

in particolare per $m = 14$ si ricava $n = 5$.

Esercizio 3. Dimostriamo l'asserto per la conversione binario–ottale (per la conversione binario–decimale si opera in modo analogo). Supponiamo di avere un numero binario di 9 bit:

$$c_8c_7c_6c_5c_4c_3c_2c_1c_0$$

il suo valore può essere scritto come:

$$8^2(c_8 * 2^2 + c_7 * 2 + c_6) + 8(c_5 * 2^2 + c_4 * 2 + c_3) + (c_2 * 2^2 + c_1 * 2 + c_0).$$

Poiché ogni espressione in parentesi è un valore compreso tra 0 e 7, tutta l'espressione può essere messa nella forma:

$$8^2 * d_2 + 8 * d_1 + d_0$$

dove d_2, d_1 e d_0 sono le cifre ottali corrispondenti alle espressioni in parentesi. Non è possibile estendere la proprietà alla conversione ottale–esadecimale perché 16 non è una potenza di 8; occorre quindi passare da ottale a binario e successivamente da binario a esadecimale.

Esercizio 4. Il volume è dato da $L * L * H/3$. Esprimendo in binario si ha:

$$\begin{aligned} &10010 * 10010 * (100 * 10010 - 1000)/11 = \\ =&10010 * 10010 * (1001000 - 1000)/11 = \\ =&10010 * 10010 * 1000000/11 = \\ =&110 * 10010 * 1000000 = \\ =&1101100000000\, m^{11}. \end{aligned}$$

Nota la scrittura di "metri cubi".

Esercizio 5. Dalla relazione $2^k \geq n + k + 1$ otteniamo $k = 3$ ed $m = 7$. Il formato della parola è $c_1c_2a_3c_4a_5a_6a_7$. I bit di controllo si ricavano con le relazioni:

$$c_1 = \text{somma mod2}(a_3a_5a_7)$$
$$c_2 = \text{somma mod2}(a_3a_6a_7)$$
$$c_4 = \text{somma mod2}(a_5a_6a_7).$$

Si ha pertanto:

Decimale	BCD	Hamming
0	0000	0000000
1	0001	1101001
2	0010	0101010
3	0011	1000011
4	0100	1001100
5	0101	0100101
6	0110	1100110
7	0111	0001111
8	1000	1110000.

Esercizio 6.

1. 18 + 6 = in ASCII diventa 0110001 0111000 0011011 0110110 0111101;
2. la trasformazione in binario fornisce due numeri: 00010010 e 00000110;
3. l'addizione fornisce il risultato: 00011000 la codifica ASCII del risultato è: 0110010 0110100.

Unità 2

Algebra Booleana

0. INTRODUZIONE

0.1. Presentazione

Come abbiamo detto più volte nell'unità precedente, i calcolatori elettronici sono realizzati mediante circuiti elementari caratterizzati da due stati di funzionamento, corrispondenti a tensione alta o bassa, a circuito aperto o chiuso, a corrente in un verso o nel verso opposto.
Per l'analisi e la sintesi di questi circuiti si utilizza uno strumento matematico chiamato Algebra Booleana, dal nome del matematico e filosofo inglese George Boole (1815, 1864). Questo tipo di algebra venne introdotto per provare analiticamente la verità o la falsità di proposizioni complesse costruite combinando, secondo le regole sintattiche della lingua inglese, proposizioni semplici di cui si conosceva, o si postulava, la verità o la falsità. L'algebra booleana era originariamente definita su un insieme costituito da due soli elementi; successivamente, è stata generalizzata per insiemi costituiti da un numero di elementi pari a una potenza di 2. Si possono quindi avere algebre a due, quattro, otto valori.
Nel 1936, lo scienziato americano Claude Shannon propose di applicare l'algebra booleana allo studio e alla progettazione dei circuiti a relè, circuiti caratterizzati dalla proprietà di poter assumere due soli stati di funzionamento (aperto e chiuso). Per lo studio di questi circuiti è sufficiente un'algebra a due valori: per questo l'algebra booleana, definita su un insieme di due elementi,è chiamata anche algebra binaria. Altra denominazione è quella di algebra di commutazione, perché i circuiti descritti da questa algebra possono commutare da uno stato all'altro. Altrimenti, in relazione alle sue origini, l'algebra booleana è chiamata anche algebra logica, e i relativi circuiti sono detti, di conseguenza, circuiti logici. Algebra booleana, algebra binaria, algebra della commutazione e algebra logica sono, in questo corso, sinonimi. Sono anche da considerarsi sinonimi "circiti di commutazione" e "circuiti logici".

0.2. Prerequisiti

Per lo studio di questa unità è necessario aver assimilato i concetti che hai studiato nell'unità precedente.

0.3. Obiettivi

Alla fine dello studio di questa unità avrai preso padronanza con l'algebra booleana e le funzioni di commutazione che giocano un ruolo fondamentale nella realizzazione dei circuiti dei calcolatori elettronici.

0.4. Parole chiave

In questa unità incontrerai molti termini nuovi. Essi sono, in ordine di apparizione, i seguenti:

algebra di commutazione	1.
AND	1.
OR	1.
NOT	1.
variabile di commutazione	1.
teorema di De Morgan	1.
legge di dualità	1.
funzione di commutazione	2.2.
tabella di verità	2.2.
mintermine	2.3.
maxtermine	2.4.
forma canonica	2.5.
semplificazione	2.6.
mappa di Karnaugh	3.1.
condizioni non specificate	3.4.
operatore universale	4.1.
NAND	4.2.
NOR	4.3.
OR esclusivo	5.
somma modulo 2	5.
anticoincidenze	5.
coincidenze	4.1.
circuiti logico	6.1.
porta logica	6.1.

1. ALGEBRA DI COMMUTAZIONE

L'algebra di commutazione è definita da un insieme di due elementi, chiamati convenzionalmente 0 e 1, e da tre operatori chiamati negazione o NOT, prodotto logico o AND, e somma logica o OR. I tre operatori sono definiti dalle seguenti regole:
NOT (è rappresentato con il segno - sopra l'elemento o con un apice):

$$\overline{1}=0 \quad \overline{0}=1 \quad \text{o anche} \quad 1'=0 \quad 0'=1$$

e si leggono "non 1" e "non 0" oppure "1 negato" e "0 negato".
AND (è rappresentato con il segno · tra due elementi):

$$0\cdot 0=0\cdot 1=1\cdot 0=0 \quad 1\cdot 1=1.$$

OR (è rappresentato con i segno + tra due elementi):

$$1+1=1+0=0+1=1 \quad 0+0=0.$$

L'operatore NOT è quindi unario perché agisce su di un singolo elemento; AND e OR sono invece operatori binari.

Domanda 1. Quanto vale $\overline{0}$?

Variabile di commutazione è una grandezza algebrica che può assumere due valori $(0,1)$. Anche sulle variabili sono definiti i tre operatori NOT, AND, OR, per i quali valgono le seguenti identità (x, y, z sono generiche variabili di commutazione; si legge "non x" o "x negato"):

$$\begin{array}{ll} x+1=1 & x\cdot 1=x \\ x+0=x & x\cdot 0=0 \\ x+\overline{x}=1 & x\cdot\overline{x}=0 \\ x+x=x & x\cdot x=x. \end{array}$$

L'ultima proprietà è chiamata idempotenza, ed esprime il fatto che il prodotto (o la somma) di più variabili uguali ha per risultato la variabile stessa.
Ognuna delle relazioni sopra viste può essere provata dando alla variabile x i due valori possibili. Ad esempio, per verificare che $x\cdot 0=0$, basta mostrare che, per $x=0$ e per $x=1$, si ha rispettivamente: $0\cdot 0=0$ e $1\cdot 0=0$.
Valgono poi le proprietà seguenti:

- commutativa:

$$x+y=y+x; \quad x\cdot y=y\cdot x;$$

- associativa:

$$x+(y+z)=(x+y)+z=x+y+z; \quad x\cdot(y\cdot z)=(x\cdot y)\cdot z=x\cdot y\cdot z;$$

– distributiva:

$$(x+y)\cdot(x+z)=x+(y\cdot z); \quad (x\cdot y)+(x\cdot z)=x\cdot(y+z)$$

nonché i teoremi dell'assorbimento:

$$x+(x\cdot y)=x \qquad x\cdot(x+y)=x$$
$$x+(\overline{x}\cdot y)=x+y \qquad x\cdot(\overline{x}+y)=x\cdot y$$

e il teorema di De Morgan:

$$\overline{x+y}=\overline{x}\cdot\overline{y}; \quad \overline{x\cdot y}=\overline{x}+\overline{y}.$$

Quest'ultimo teorema è molto importante, perché permette di esprimere l'operatore OR in funzione degli operatori AND e NOT, nonché l'operatore AND in funzione degli operatori OR e NOT: torneremo in seguito su questo punto.

Domanda 2. Il teorema di De Morgan vale per un numero qualsiasi di variabili; come si applica allora all'espressione: $\overline{x+y+z+w}$?

Dalle proprietà degli operatori si ricava un principio generale. La legge di dualità dell'algebra booleana:

> ogni identità e ogni proprietà dell'algebra booleana resta valida se si scambiano tra di loro gli operatori AND e OR e gli elementi 0 e 1.

Analogamente alla convenzione usata nell'algebra dei numeri reali, si usa la regola di precedenza dell'operatore prodotto (AND) rispetto all'operatore somma (OR), in modo da poter eliminare alcune parentesi. È anche di norma sottinteso il simbolo di prodotto logico, effettuando una semplice giustapposizione dei termini da connettere mediante l'operatore AND.

Esempio 1. La relazione $(x\cdot y)+(x\cdot z)=x\cdot(y+z)$, con le convenzioni predette, si scrive $xy+xz=x(y+z)$, proprio come nell'algebra dei numeri reali.

2. FUNZIONI DI COMMUTAZIONE

2.1. Introduzione

Una funzione di commutazione $y=f(x)$ è una funzione in cui il dominio e il codominio sono l'insieme $(0,1)$.
Poiché tanto il dominio che il codominio hanno un numero finito di elementi (qui due elementi), anche il numero di funzioni di commutazione y_i di una variabile x deve essere finito. Più precisamente, queste funzioni sono quattro, come puoi vedere nella tabella 1.

x	y_1	y_2	y_3	y_4
0	0	0	1	1
1	0	1	0	1

Tabella 1. *Funzioni di una variabile.*

Nota che y_1 è la funzione identicamente uguale a 0; y_4 è la funzione identicamente uguale a 1; y_2 è la funzione $y=x$; y_3 è la funzione $y=\overline{x}$.
Similmente a quanto abbiamo fatto per unavariabile, possiamo definire le funzioni di commutazione di più variabili. Una funzione di commutazione di n variabili $y=f(x_1,x_2,\ldots,x_n)$ è una funzione in cui il dominio è costituito da tutte e sole le n–ple $(x_1,x_2,\ldots,x_n)$ di elementi dell'insieme $(0,1)^n$ ed il cui codominio è l'insieme $(0,1)$. Anche in questo caso, la totalità delle funzioni definibili è un numero finito. Infatti, il numero delle n–ple differenti è 2^n, che corrispondono agli elementi del dominio $(0,1)^n$.
Poiché una funzione $y=f(x_1,x_2,\ldots,x_n)$ è una applicazione di $(0,1)^n$ su $(0,1)$, le possibili scelte sono le disposizioni di due elementi $(0,1)$ su 2^n posti e quindi pari a:

$$2^{2^n}$$

Allo stesso modo, il numero di colonne diverse di una schedina del Totocalcio deriva dalla applicazione di un dominio di 13 elementi (le partite) su un codominio di tre elementi $(1,X,2)$ e quindi vale 3^{13}.

Esercizio 1. Scrivi, come nella tabella 1, tutte le funzioni di commutazione di due variabili.

2.2. Tabelle di verità

Come abbiamo già visto, i valori di n variabili di commutazione $x_1,x_2,\ldots,x_n$ generano 2^n configurazioni diverse degli elementi 0 e 1. La scrittura $x_1,x_2,\ldots,x_n=01\ldots0$ indica la configurazione in cui $x_1=0$; $x_2=1;\ldots;x_n=0$.
Introduciamo ora un vettore X che ha come componenti i valori delle n variabili $x_1,x_2,\ldots,x_n$, ossia $X=<x_1,x_2,\ldots,x_n>$, e interpretiamo questo vettore come un numero binario di n bit, in grado quindi di assumere tutti i valori compresi tra

0 e $2^n - 1$. Per ciascuna delle possibili funzioni di commutazione si può allora costruire una tabella che riporti, per ogni valore di X (interpretato come numero binario), il corrispondente valore della funzione $y = f(X)$. La tabella ottenuta si chiama tabella di verità della funzione y, e ne costituisce una prima efficace rappresentazione.

Esempio 2. Qui di seguito è riportata la tabella di verità per una delle 256 possibili funzioni di tre variabili.

N	X_1	X_2	X_3	Y	N	X_1	X_2	X_3	Y
0	0	0	0	1	4	1	0	0	0
1	0	0	1	0	5	1	0	1	1
2	0	1	0	0	6	1	1	0	1
3	0	1	1	1	7	1	1	1	0

Facciamo notare che, nella prima e nella sesta colonna della tabella le configurazioni delle tre variabili sono state interpretate come se fossero numeri binari. Questo passaggio dall'uno all'altro significato dei simboli 0 e 1, anche se può ingenerare qualche confusione nei principianti, è molto comune, e serve a semplificare radicalmente molte notazioni e trattazioni.

Esercizio 2. Scrivi la tabella di verità per una funzione di tre variabili che vale 1 in corrispondenza delle configurazioni che hanno un numero dispari di 1, e vale 0 in corrispondenza di quelle che ne hanno un numero pari.

2.3. Mintermine

Abbiamo visto che un vettore X di n variabili può assumere 2^n valori distinti. È utile poter indicare queste configurazioni con dei numeri decimali, interpretando le n–ple dei valori di X come numeri binari che possono essere espresse anche in decimale.

Esempio 3. Le quattro configurazioni possibili del vettore di due variabili x_1 ed x_2 sono 00; 01; 10, 11 che possiamo anche indicare con 0, 1, 2, 3.

Definiamo mintermine delle n variabili $x_1, x_2, \ldots, x_n$ un generico prodotto di tutte le variabili $x_1, x_2, \ldots, x_n$, dirette o negate; poiché i prodotti diversi che si possono realizzare sono pari al numero di configurazioni delle n variabili è possibile associare ad ogni configurazione un mintermine in modo tale che il suo valore vale 1 quando le n variabili assumono la configurazione ad esso associata. Pertanto un mintermine può essere indicato con il simbolo m_i, essendo i il corrispondente valore in decimale del vettore X definito nel precedente paragrafo, ed è espresso negando la variabile x_j se il bit j–esimo di X è 0.

Esempio 4. Il mintermine m_6 delle tre variabili x_1, x_2, x_3 è il prodotto corrispondente al valore $X = 6$, cioè alla configurazione 110, ed è pertanto espresso con:

$$m_6 = x_1 x_2 \overline{x}_3$$

ed infatti il prodotto delle tre variabili $x_1 x_2 \overline{x}_3$ è pari ad uno in corrispondenza alla configurazione 110.

Dalla tabella di verità di una funzione di commutazione $y = f(x_1, x_2, \ldots, x_n)$, e dalla definizione di mintermine, si deduce facilmente che la funzione y può essere scritta in forma analitica come somma logica dei mintermini corrispondenti a quei valori di X per cui la funzione vale 1.
La rappresentazione analitica è più concisa e pratica di quella mediante tabella di verità; le due rappresentazioni sono comunque deducibili l'una dall'altra, come mostra l'esempio seguente.

Esempio 5. La funzione rappresentata nella tabella di verità dell'esempio 2 vale 1 in corrispondenza ai valori $X = 0$, $X = 3$, $X = 5$, $X = 6$ cioè in corrispondenza ai mintermini m_0, m_3, m_5, m_6, e poiché:

$$m_0 = \overline{x}_1 \overline{x}_2 \overline{x}_3$$

$$m_3 = \overline{x}_1 x_2 x_3$$

$$m_5 = x_1 \overline{x}_2 x_3$$

$$m_6 = x_1 x_2 \overline{x}_3$$

la funzione si scrive in forma analitica come:

$$y = \overline{x}_1 \overline{x}_2 \overline{x}_3 + \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 x_3 + x_1 x_2 \overline{x}_3.$$

Viceversa, partendo dall'espressione analitica della funzione y, si passa alla tabella di verità:

- associando ad ogni mintermine m_i della somma il corrispondente valore i di X;
- scrivendo 1 nelle righe i della tabella di verità;
- scrivendo 0 nelle restanti righe.

Domanda 3. Qual è la tabella di verità corrispondente alla funzione $y = x_1 \overline{x}_2 + \overline{x}_1 x_2$?

2.4. Maxtermine

Definiamo maxtermine delle n variabili $x_1, x_2, \ldots, x_n$ la somma di tutte le variabili, dirette o negate. Un maxtermine si indica con il simbolo M_i, dove i è il corrispondente valore del vettore X definito nel paragrafo 2.2., ed è espresso negando la variabile x_j se il bit j–esimo di X è 1.

Esempio 6. Il maxtermine M_6 delle variabili x_1, x_2, x_3 corrisponde al valore $X = 6$, cioè alla configurazione 110 delle variabili x_1, x_2, x_3; si esprime pertanto con:

$$M_6 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3.$$

Una seconda forma analitica di una funzione di commutazione $y = f(x_1, x_2, \ldots, x_n)$ si ottiene scrivendo il prodotto dei maxtermini in corrispondenza dei quali la funzione y è 0 (ti risparmiamo la dimostrazione di questa regola, che deriva dalla legge di dualità che abbiamo visto nella prima sezione di questa unità).

Esempio 7. La funzione rappresentata nella tabella di verità dell'esempio 2 è 0 in corrispondenza dei maxtermini:

$$M_1, \quad M_2, \quad M_4, \quad M_7$$

i quali corrispondono, rispettivamente, ai valori:

$$X_1 = 001, \quad X_2 = 010, \quad X_4 = 100, \quad X_7 = 111.$$

Si ha pertanto:

$$M_1 = x_1 + x_2 + \overline{x}_3$$

$$M_2 = x_1 + \overline{x}_2 + x_3$$

$$M_4 = \overline{x}_1 + x_2 + x_3$$

$$M_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3.$$

La y si scrive quindi:

$$y = (x_1 + x_2 + \overline{x}_3)(x_1 + \overline{x}_2 + x_3)(\overline{x}_1 + x_2 + x_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3).$$

Esercizio 3. Prova a scrivere la funzione $ab' + a'b$ nella forma "prodotto di maxtermini".

2.5. Forme canoniche di una funzione

Come abbiamo visto nel paragrafo precedente, ogni funzione di commutazione può essere espressa analiticamente come somma di mintermini o prodotto di maxtermini. Queste due forme di una funzione, ricavabili direttamente dalla sua tabella di verità, si chiamano rispettivamente forma canonica in somma di prodotti e forma canonica in prodotto di somme.
In generale la rappresentazione di una generica funzione di n variabili si può esprimere sinteticamente con due forme canoniche che si chiamano rispettivamente:

1) forma canonica in somma di mintermini;
2) forma canonica in prodotto di maxtermini.

La prima forma canonica assume la forma:

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

dove $f(i)$ indica il valore assunto dalla y in corrispondenza di X_i e il simbolo di sommatoria va inteso come somma logica.
La seconda forma canonica si rappresenta come:

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

dove $f(i)$ ha lo stesso significato ed il simbolo di produttoria va inteso come prodotto logico.
Nella prima forma canonica, per una particolare funzione, si considereranno solo quei mintermini in corrispondenza dei quali $f(i) = 1$, mentre nella seconda forma canonica si considereranno solo quei maxtermini in corrispondenza dei quali $f(i) = 0$. Infatti:

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

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

Spesso si usa una forma abbreviata delle forme canoniche, chiamata "rappresentazione decimale", nella quale viene indicato sinteticamente la sommatoria (la produttoria) degli indici dei mintermini (dei maxtermini).

Esempio 8. La funzione rappresentata nella tabella di verità dell'esempio 2 può essere scritta in ognuna delle due forme decimali:

$$y = \sum(0, 3, 5, 6)$$

$$y = \prod(1, 2, 4, 7)$$

Esercizio 4. Applicando quanto abbiamo detto, scrivi ora le rappresentazioni decimali della funzione avente la seguente tabella di verità:

X	x_1	x_2	x_3	y
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

2.6. Forme semplificate di una funzione

Abbiamo finora mostrato tre diverse rappresentazioni delle funzioni di commutazione, una tabellare e due analitiche (che abbiamo chiamato "canoniche"). Abbiamo inoltre espresso in forma "decimale" le due rappresentazioni canoniche. Generalmente esistono anche, accanto alle forme canoniche di una funzione, altre forme analitiche equivalenti (nel senso che corrispondono alla stessa tabella di verità) della funzione stessa. Queste forme non canoniche si chiamano forme semplificate e sono caratterizzate dal fatto di contenere meno variabili, o meno operatori, o meno variabili e meno operatori complessivamente, delle forme canoniche. Tra tutte le forme semplificate sono particolarmente importanti quelle espresse sotto forma di somma di prodotti o di prodotto di somme.
I prodotti che non contengono tutte le variabili si chiamano implicanti della funzione, mentre le somme che non contengono tutte le variabili si chiamano implicati.
Si chiama semplificazione ogni procedimento che porta ad una forma semplificata di una funzione. Esistono diversi metodi di semplificazione; tra tutti, descriveremo brevemente un metodo analitico e – nella prossima sezione – un metodo grafico (le mappe di Karnaugh).
Osservazione importante: potrebbe venire in mente di cercare la "forma minima" fra tutte le forme semplificate. In effetti, a questa ricerca sono state dedicate, in passato, molte energie. Il problema – che veniva affrontato con metodi che dipendevano molto dal tipo di definizione data alla "forma minima" – ha oggi minore importanza pratica, rispetto al passato, per il basso costo degli elementi con cui si realizzano in circuiti le funzioni di commutazione.
Il metodo analitico di semplificazione consiste nell'operare diverse trasformazioni di una funzione, sfruttando le proprietà riportate nella sezione 1.
Per l'applicazione del metodo, si parte da una espressione qualsiasi della funzione da semplificare, preferibilmente dalla sua forma canonica come somma di prodotti. Il metodo è semplice, e particolarmente conveniente quando l'espressione da semplificare contiene pochi termini e molte variabili.

Esempio 9. Semplificare la seguente funzione scritta in forma canonica come somma di mintermini:

$$f = xyzw + xyz\overline{w} + xy\overline{z}\,\overline{w} + xy\overline{z}w + \overline{x}yzw.$$

Semplifichiamo i primi due mintermini:

$$xyzw + xyz\overline{w} = xyz.$$

Con la stessa regola il primo termine può essere semplificato con il quarto, in accordo con la proprietà dell'idempotenza (vedi la sezione 1) secondo la quale un termine può essere ripetuto un numero qualsiasi di volte; ed ancora il primo termine può essere semplificato con il quinto, e il secondo con il terzo:

$$xyzw + xy\overline{z}w = xyw$$

$$xyzw + \overline{x}yzw = yzw$$

$$xyz\overline{w} + xy\overline{z}\,\overline{w} = xy\overline{w}.$$

Possiamo allora scrivere:

$$f = xyz + xyw + yzw + xy\overline{w}.$$

Semplifichiamo ora il secondo termine con l'ultimo ottenendo xy e per la proprietà dell'assorbimento ($a + ab = a$) otteniamo l'espressione semplificata come somma di due implicanti:

$$f = xy + yzw.$$

Esercizio 5. Prova a semplificare la funzione:

$$y = \prod(2, 4, 5, 6).$$

Come hai notato, la regola di semplificazione è banale; purtroppo non esiste un procedimento lineare che guidi nella semplificazione di una funzione. Per ottenere una "buona" forma semplificata ci si affida all'abilità, all'intuito e, perché no, alla fortuna del semplificatore
Un metodo di semplificazione più sicuro è quello di Karnaugh, che vedremo nella prossima sezione; ha però l'inconveniente di essere applicabile solo a funzioni con un numero limitato di variabili (al più, cinque).
Osservazione importante: abbiamo detto che ogni forma semplificata di una funzione ha la stessa tabella di verità della corrispondente forma canonica. Una conseguenza di quest'affermazione è che tutte le forme semplificate corrispondono alla stessa forma canonica.
Nel paragrafo seguente, vedremo come passare da una forma semplificata alla forma canonica in prodotto di somme di una stessa funzione.

2.7. Passaggio da una forma generica a quella canonica

Scrivere la tabella di verità di una funzione in forma semplificata presuppone il passaggio dalla somma di implicanti che rappresenta la funzione stessa, alla forma canonica corrispondente. Questo passaggio, che è in ultima analisi una semplificazione alla rovescia, consiste nell'applicazione della proprietà:

$$xy = xy(z + \overline{z}) = xyz + xy\overline{z}.$$

Si passa, in questo modo, da un prodotto di k variabili a due prodotti di $k + 1$ variabili. Il procedimento va ripetuto su tutti i termini della funzione semplificata, finché non si ottengono tutti e soli prodotti di n variabili, cioè mintermini. Eventuali mintermini ottenuti più volte, vanno considerati una volta sola.

Esempio 10. La funzione $f = xy + yzw$ è funzione delle quattro variabili x, y, w, z. Per scriverla in forma canonica, dobbiamo ottenere tutti mintermini, cioè i prodotti delle quattro variabili x, y, w, z.
Dal termine xy si ottiene:

$$xy = xy(z + \overline{z}) = xyz + xy\overline{z}$$

e, successivamente:

$$xyz + xy\overline{z} = xyz(w + \overline{w}) + xy\overline{z}(w + \overline{w}) = xyzw + xyz\overline{w} + xy\overline{z}w + xy\overline{z}\,\overline{w}.$$

Dal termine yzw si ottiene:

$$ywz = ywz(x + \overline{x}) = xyzw + \overline{x}yzw.$$

Si ha pertanto:

$$f = xyzw + xyz\overline{w} + xy\overline{z}w + \overline{x}yzw + xy\overline{z}\,\overline{w}.$$

Domanda 4. Qual è la tabella di verità della funzione $y = x + wz$?

3. MAPPE DI KARNAUGH

3.1. Introduzione

Le mappe di Karnaugh sono delle tabelle che permettono la rappresentazione e la semplificazione delle funzioni di commutazione fino a quattro variabili. È possibile usarle, con qualche difficoltà, anche per funzioni di cinque variabili. Esistono anche le mappe per sei variabili, ma il loro uso è piuttosto complicato.
Le mappe di Karnaugh per le funzioni di 2, 3, 4, 5 variabili sono divise in tante caselle (o "celle") quanti sono i corrispondenti mintermini (4, 8, 16, 32).
La mappa di Karnaugh per due variabili, riportata nella figura 1, è un quadrato diviso in quattro celle. Ogni cella è individuata dai valori assunti dalle due variabili e il suo contenuto è il valore assunto dalla funzione per la configurazione delle variabili di ingresso corrispondente. In altri termini le variabili di ingresso indirizzano le celle e il loro contenuto è il corrispondente valore della funzione $f(x, y)$.

y \ x	0	1	
0	f (00)	f (10)	0
1	f (01)	f (11)	1

Figura 1. *Mappa per due variabili.*

Nota che le caselle adiacenti corrispondono a configurazioni delle variabili di ingresso che differiscono di un bit; per esempio le caselle contenenti $f(10)$ e $f(11)$ differiscono solo per il valore assunto dalla y.
La mappa di Karnaugh per tre variabili, riportata nella figura 2, è ottenuta per rotazione della mappa per due variabili intorno ad un asse verticale che individua alla sua sinistra le caselle di indirizzo $x = 0$ e alla sua destra le caselle di indirizzo $x = 1$.

y \ xz	00	10	11	01
0	f (000)	f (100)	f (110)	f (010)
1	f (001)	f (101)	f (111)	f (101)

Figura 2. *Mappa di Karnaugh della funzione* $f(x, z, y)$.

Puoi notare che le caselle adiacenti corrispondono a configurazioni delle variabili che differiscono di un bit ed occorre sottolineare che anche le caselle sulle due colonne estreme sono da considerarsi adiacenti, come se la mappa fosse originariamente su un cilindro che è stato spianato.

La mappa di Karnaugh per quattro variabili è ottenua in modo analogo facendo ruotare quella da tre intorno ad un asse orizzontale che individua al di sopra le caselle di indirizzo $w = 0$ e al di sotto le caselle di indirizzo $w = 1$.

y w \ x z	00	10	11	01
00	f (0000)			f (0100)
10				
11			f (1111)	
01	f (0001)			f (0101)

Figura 3. *Mappa di Karnaugh della funzione* $f(x, z, y, w)$.

Nella mappa di quattro variabili le caselle adiacenti sono anche quelle sulle righe estreme, oltre che quelle sulle colonne estreme, come si può notare dalla figura 3.

z w \ x y	00	01	11	10
00	f (0000)			f (1000)
01				
11			f (1111)	
10	f (0010)			f (1010)

Figura 4. *Una diversa mappa della funzione* $f(x, y, z, w)$.

Naturalmente l'ordine con cui vengono usate le variabili per indirizzare le colonne e le righe può essere cambiato in modo arbitrario, così come è possibile cambiare la codifica delle colonne e delle righe rispettando però l'adiacenza tra due codifiche consecutive.

In figura 4 è riportata una mappa per quattro variabili in cui si sono scambiate le variabili y e z ed in cui le codifiche delle righe e delle colonne sono diventate: 00; 01; 11; 10.

3.2. Rappresentazione di una funzione sulle mappe

Come abbiamo detto più sopra, le mappe di Karnaugh sono anzitutto un metodo per rappresentare le funzioni di commutazione Questo tipo di rappresentazione è particolarmente semplice a partire dalla tabella di verità di una funzione f, in quanto basta scrivere 1 nelle celle le cui coordinate corrispondono alle configurazioni delle variabili x_i per cui la f vale 1 (vedi il paragrafo 2.2.) e 0 nelle rimanenti caselle.

Nell'uso pratico si lasciano vuote le caselle in cui andrebbe messo 0, intendendolo sottointeso; in questa unità, tuttavia riempiremo tutte le caselle di una mappa per maggior efficacia e chiarezza.

Esempio 11. Rappresenta sulla mappa di Karnaugh la funzione di tre variabili avente la seguente tabella di verità:

X	x_1	x_2	x_3	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Scrivendo 1 nelle celle le cui coordinate corrispondono alle configurazioni di $x_1x_2x_3$ per cui $f = 1$, si ha la rappresentazione indicata nella figura 5.

Nell'uso pratico si marcano con 1 le caselle in cui la funzione vale 1 e si lasciano vuote le rimanenti. In questa unità, per maggior efficacia riempiremo tutte le caselle, ma in seguito useremo la pratica consolidata di lasciare vuote le caselle in cui andrebbe 0.

x3 \ x1 x2	00	01	11	10
0	1	0	0	1
1	0	1	1	1

Figura 5. *Rappresentazione di una funzione di tre variabili sulla mappa di Karnaugh.*

Abbiamo visto come rappresentare sulle mappe di Karnaugh una funzione di cui è data la tabella di verità. È possibile comunque rappresentare sulle mappe una funzione data in forma analitica, come somma di prodotti, senza passare prima per la relativa tabella di verità. In pratica:

- se la funzione è data come somma di mintermini, basta scrivere 1 in tutte le celle corrispondenti ai mintermini della somma;
- se la funzione è data in forma semplificata, basta scrivere 1 in tutte le celle corrispondenti ai mintermini contenuti in ogni prodotto implicante.

Ad esempio, l'implicante $\bar{x}z$, in una funzione delle tre variabili x, y, z, corrisponde alla somma di due mintermini: $\bar{x}yz + \bar{x}\bar{y}z$ (vedi il paragrafo 2.5.); su una mappa di tre variabili (vedi la figura 2) comprende quindi le celle di coordinate $xyz = 001$ e $xyz = 011$. Nota che queste celle sono tutte e sole quelle che hanno coordinate $x = 0$ e $z = 1$.
Osservazione importante: abbiamo visto come si rappresenta, sulle mappe di Karnaugh, una funzione espressa in "somma di prodotti". Le funzioni nella forma "prodotto di somme" possono essere rappresentate dopo averle riportate nella prima forma. Di norma, comunque, la rappresentazione sulle mappe si effettua a partire dalla tabella di verità.
Una proprietà caratteristica delle mappe di Karnaugh è che un prodotto di variabili che non è un mintermine (ad esempio, xz su una mappa delle tre variabili x, y, z) occupa sempre celle adiacenti della mappa.
Per chiarire il concetto, abbiamo riportato sulla mappa di Karnaugh per tre variabili della figura 6 alcuni prodotti di due variabili. Nota che ognuno di essi occupa due celle adiacenti, che corrispondono ai due mintermini in cui la terza variabile appare una volta diretta e una volta negata.

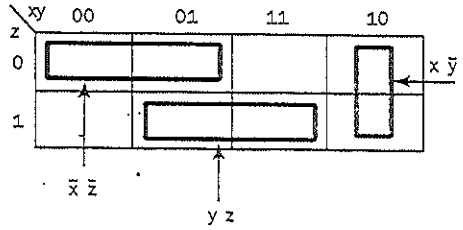


Figura 6. *Prodotti di due variabili sulla mappa di Karnaugh per tre variabili.*

Nella figura 7 sono riportati su una mappa per quattro variabili, alcuni prodotti di tre variabili, ottenuti per unione di due celle adiacenti, nonché alcuni prodotti di due variabili, ottenuti per unione di quattro celle adiacenti.

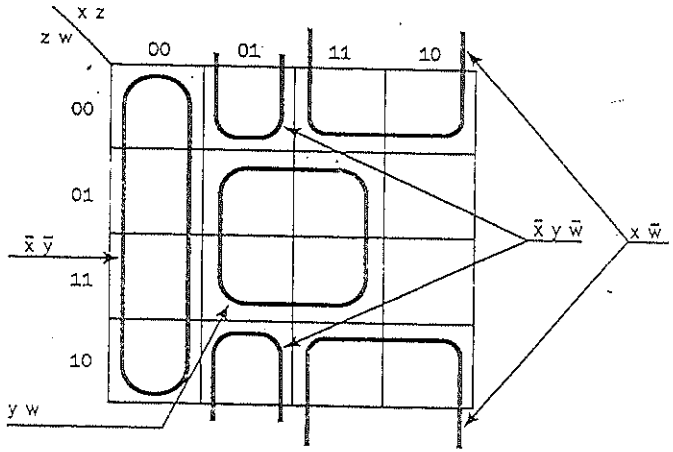


Figura 7. *Prodotti di due e tre variabili sulla mappa di Karnaugh per quattro variabili.*

Domanda 5. Quale funzione è rappresentata sulla mappa di Karnaugh di figura 8?

z \ x y	00	01	11	10
1	1	1	0	1
0	0	0	1	0

Figura 8. *Mappa di Karnaugh per una funzione di tre variabili.*

Esercizio 6. Rappresenta su una mappa di Karnaugh la funzione:

$$f = a + bc + \bar{b}\bar{c}.$$

3.3. Semplificazione di una funzione

La semplificazione di una funzione rappresentata sulle mappe di Karnaugh consiste nel raggruppare le celle segnate con 1 nel minor numero di insiemi di 8, 4, 2 celle adiacenti, ed eventualmente di celle isolate, in modo da prenderle tutte almeno una volta. L'espressione minima della funzione si ottiene sommando i prodotti corrispondenti agli insiemi così formati.

L'efficacia del metodo è subordinata alla capacità dell'occhio umano di riconoscere delle configurazioni, che sono particolarmente evidenti sulle mappe di Karnaugh (almeno dopo un po' di esercizio.).

Esempio 12. Semplificare la funzione rappresentata nella figura 9.a.

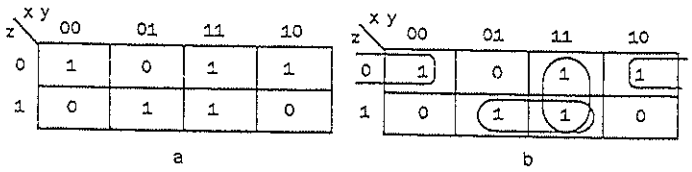


Figura 9. *Mappa di una funzione di tre variabili.*

La funzione può essere semplificata raggruppandone i mintermini in tre insiemi di due caselle, nel modo evidenziato nella figura 9.b. Si ha così l'espressione minima in somma di prodotti:

$$f = \overline{y}\,\overline{z} + xy + yz.$$

La stessa funzione può essere semplificata anche in modo diverso, dando luogo ad una espressone minima della y equivalente a quella data:

$$f = x\overline{z} + xy + \overline{y}\,\overline{z}.$$

Domanda 6. Come è stata ottenuta la seconda forma della funzione y ?

La semplificazione di una funzione a quattro variabili non presenta difficoltà molto maggiori, come mostra l'esempio seguente.

Esempio 13. Semplificare la funzione di quattro variabili rappresentata nella figura 10.a.

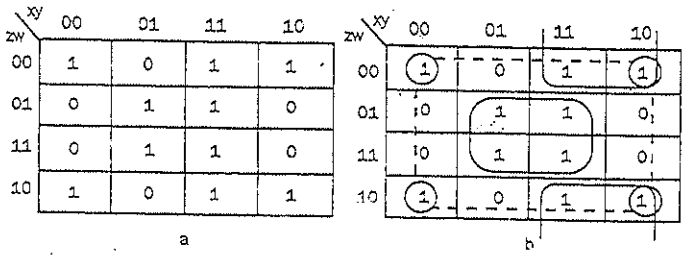


Figura 10. *Mappa di una funzione di quattro variabili.*

La funzione può essere semplificata formando tre insiemi di quattro celle, nel modo evidenziato nella figura 10.b. La forma minima come somma di prodotti

è perciò:

$$f = x\overline{w} + yw + \overline{y}\,\overline{w}.$$

Come puoi vedere facilmente, la funzione non dipende da z, ma solo dalle tre variabili x, z, w (le semplificazioni riservano, talvolta, sorprese del genere!)

Domanda 7. Come si rappresenta sulla mappa di Karnaugh per tre variabili la funzione rappresentata sulla mappa di figura 10.a?

3.4. Funzioni parzialmente specificate

Nella definizione di una funzione di commutazione y, possono talvolta non essere specificati i valori (0 o 1) assunti dalla y stessa in corrispondenza di un certo numero di configurazioni delle variabili $x_1, x_2, \ldots, x_n$. Dette configurazioni si chiamano "condizioni non specificate" (in inglese, don't care condictions – d.c.c.). In pratica, come vedremo spesso nelle prossime unità, le condizioni non specificate si presentano molto frequentemente nelle funzioni logiche, e sono importanti perché permettono forti semplificazioni nei corrispondenti circuiti.
Sulla tabella di verità – e sulla mappa di Karnaugh – le condizioni non specificate si indicano con un trattino.

Esempio 14. Nelle figure 11 e 12 è rappresentata, sulla tabella di verità e sulla mappa di Karnaugh, una funzione di commutazione f in cui esistono due condizioni non specificate: il valore della f non è infatti definito in corrispondenza ai mintermini m_3 e m_5:

X	x	y	z	f
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	–
4	1	0	0	1
5	1	0	1	–
6	1	1	0	0
7	1	1	1	1

Figura 11. *Tabella di verità per una funzione parzialmente specificata.*

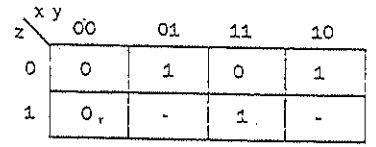


Figura 12. *Mappa di una funzione parzialmente specificata.*

Nella rappresentazione in forma canonica delle funzioni parzialmente specificate, i mintermini corrispondenti alle d.c.c. si indicano tra parentesi. Se si usa invece la rappresentazione come sommatoria dei valori decimali dei mintermini, le d.c.c. si indicano con una seconda sommatoria con indice 0/1, ad indicare che la y può assumere indifferentemente il valore 0 o il valore 1. La funzione considerata nell'esempio 14 può quindi rappresentarsi in una delle forme equivalenti:

$$f = \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz + (\bar{x}yz + x\bar{y}z)$$

oppure:

$$f = \sum(2,4,7) + \sum_{0/1}(3,5).$$

Esercizio 7. Prova a disegnare la mappa di Karnaugh della funzione di quattro variabili:

$$f = \sum(1,5,6,9) + \sum_{0/1}(7,11,13).$$

Le condizioni non specificate vanno usate, se conviene, come ulteriori valori 1, in modo da semplificare ulteriormente l'espressione della funzione. Più precisamente, assegneremo il valore 1 a quelle celle contenenti d.c.c. che ci permettono di formare insiemi corrispondenti a prodotti che altrimenti non esisterebbero. L'esempio che segue chiarirà quanto detto.

Esempio 15. Semplificare la funzione di quattro variabili:

$$f = \sum(0,5,10,15) + \sum_{0/1}(1,3,4,11,13,14)$$

La mappa di Karnaugh della funzione è rappresentata nella figura 13.a. In questo caso conviene assegnare il valore 1 a quatro delle celle d.c.c., e il valore 0 alle altre, in modo da formare i prodotti e (come mostrato nella figura 13.b.).

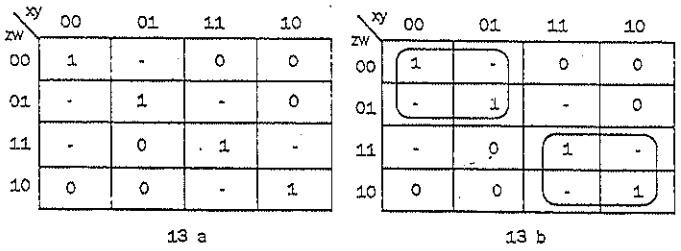


Figura 13. *Mappa di una funzione con d.c.c.*

Poiché i due implicanti individuati sulla mappa di figura 13.b. coprono tutti i mintermini della funzione, si ha:

$$f = xz + \bar{x}\bar{z}.$$

Nota che se non fossero state considerate le d.c.c., la forma minima della y sarebbe stata molto più complessa dovendo essere rappresentata dalla somma dei quattro mintermini che non sono semplificabili in quanto non adiacenti.

Domanda 8. Qual è la forma minima come somma di prodotti della funzione di tre variabili:

$$f = \prod(3,6) + \prod_{0/1}(5,7)\,?$$

Il simbolo $\prod_{0/1}$, analogamente al simbolo $\sum_{0/1}$, indica i maxtermini corrispondenti a d.c.c.

3.5. Osservazioni finali

Come abbiamo visto, le mappe di Karnaugh sono uno strumento semplice e potente per ottenere la forma minima come somma di prodotti di una funzione, completamente o parzialmente specificata. Prima di lasciare l'argomento, riteniamo comunque necessario aggiungere due note. La prima riguarda le mappe di Karnaugh a cinque variabili; la seconda un utile accorgimento per rendere immediata la rappresentazione delle funzioni sulle mappe (che è il primo passo della semplificazione).

Nota 1. Mappe di Karnaugh per cinque variabili.
Le mappe di Karnaugh per cinque variabili sono formate da due mappe per quattro variabili affiancate. Sono da considerarsi adiacenti, oltre a quelli già visti, tutti i blocchi che è possibile formare con celle occupanti le stesse posizioni sulle due mappe. Nella figura 14 sono indicati alcuni gruppi di celle adiacenti.

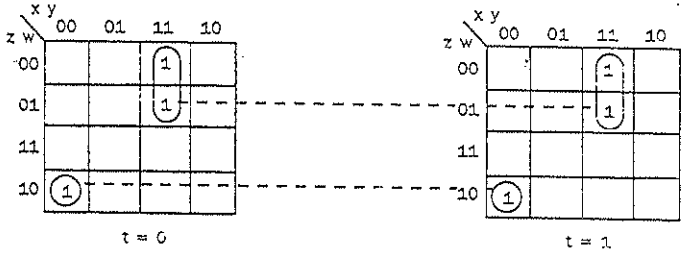


Figura 14. *Mappa di Karnaugh per cinque variabili.*

Esercizio 8. Semplificare la funzione di cinque variabili:

$$f = \sum(7,9,15,16,17,18,19,20,21,22,23,31).$$

Nota 2.
Come abbiamo visto ripetutamente, le funzioni di commutazione vengono spesso espresse con la sommatoria dei numeri decimali corrispondenti ai relativi mintermini. Allo stesso modo, si possono numerare in decimale le celle delle mappe di Karnaugh, in modo da rappresentare immediatamente le funzioni date nella forma predetta (vedi la figura 15).

z \ x y	00	01	11	10
0	0	2	6	4
1	1	3	7	5

zw \ x y	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

zw \ x y	00	01	11	10
00	0	8	24	16
01	2	10	26	18
11	6	14	30	22
10	4	12	28	20

t = 0

zw \ x y	00	01	11	10
00	1	9	25	17
01	3	11	27	19
11	7	15	31	23
10	5	13	29	21

t = 1

Figura 15. *Valori decimali delle caselle delle mappe di Karnaugh a 3, 4 e 5 variabili.*

3.6. Espressione di una funzione in prodotto di somme

Come avviene per le forme canoniche, anche per le forme semplificate delle funzioni di commutazione si può arrivare facilmente ad espressioni del tipo "prodotto di somme".
Cerchiamo di ripercorrere il cammino già fatto. Ogni funzione di commutazione y si può scrivere come somma dei mintermini corrispondenti ai valori delle variabili per cui si ha $y = 1$. Da questo deriva che la funzione negata y (che è 0 per i valori delle variabili per cui $y = 1$, e viceversa) si può scrivere come somma dei mintermini corrispondenti ai valori delle variabili per cui si ha $y = 0$. Per ottenere una forma semplificata come prodotto di somme, si segue allora il seguente procedimento:

- si prendono gli 0 della funzione f (si considera cioè la funzione $\overline{f}$);

- si ricava la forma semplificata della $\overline{f}$, come somma di prodotti;
- si applica il teorema di De Morgan alla $\overline{f}$, ricavando la f nella forma voluta.

La terza regola richiede un chiarimento. Quando si realizza $\overline{f}$ come somma di prodotti si ottiene una espressione del tipo:

$$\overline{f} = \sum_i P_i$$

per cui negando primo e secondo membro ed applicando al secondo membro il teorema di De Morgan otteniamo:

$$\overline{\overline{f}} = f = \overline{\sum_i P_i} = \prod_i \overline{P}_i.$$

È bene osservare che, qualora non esistano vincoli sulla forma della funzione, conviene ricavare la forma ridotta in prodotto di somme quando gli 1 della f sono più di $2^n - 1$, cioè più degli 0, essendo n il numero delle variabili. Un esempio chiarirà meglio il procedimento illustrato.

Esempio 16. Semplificare la funzione:

$$f = \sum(0, 1, 2, 3, 5, 7, 8, 9, 10, 11)$$

ricavando la forma minima come prodotto di somme.
La mappa di Karnaugh per la f è riportata nella figura 16.

zw \ x y	00	01	11	10
00	1	0	0	1
01	1	1	0	1
11	1	1	0	1
10	1	0	0	1

Figura 16. *Mappa della funzione da semplificare*

L'espressione minima della funzione y, ottenuta considerando i blocchi formati con le caselle con 0 è:

$$\overline{f} = xy + y\overline{w}$$

per il teorema di De Morgan, si ha quindi:

$$f = \overline{\overline{f}} = \overline{xy + y\overline{w}} = \overline{xy}\,\overline{y\overline{w}} = (\overline{x} + \overline{y})(\overline{y} + w).$$

Esercizio 9. Scrivi la forma minima come prodotto di somme della funzione:

$$f = \sum(1, 3, 4, 6, 7, 9, 10, 11, 12, 13, 14, 15).$$

4. OPERATORI UNIVERSALI

4.1. Introduzione

Tutte le forme analitiche (canoniche o semplificate) con cui finora si è rappresentata una funzione di commutazione, contengono di norma i tre operatori NOT, AND, OR.
Ci si può chiedere se non sia possibile esprimere una funzione definita in una tabella di verità con un insieme di operatori diversi da quelli utilizzati finora o, al limite, se non esista qualche operatore che, da solo, permetta di esprimere ogni funzione. Dal punto di vista matematico, questi problemi sono molto interessanti; da quello tecnico lo sono nella misura in cui esistono elementi che realizzano operazioni diverse da quelle relative agli operatori NOT, AND, OR. In effetti esistono due operatori (NAND e NOR) che realizzano qualsiasi funzione logica e sono detti, per questo motivo, operatori universali; questi operatori possono, in più, essere implementati da semplici circuiti elettronici.
Osservazione importante: si è visto come ogni funzione logica possa essere espressa con gli operatori NOT, AND, OR. Applicando il teorema di De Morgan si può dimostrare che l'insieme dei tre operatori è sovrabbondante. Si può esprimere ogni funzione con l'operatore NOT ed uno solo degli operatori AND o OR. Infatti, per eliminare l'operatore OR o l'operatore AND da una generica espressione in somma di prodotti contenente i tre operatori, basta effettuare una doppia negazione dell'espressione stessa, applicare il teorema di De Morgan, ed eliminare o l'operatore OR o l'operatore AND.

Esempio 17. La funzione:

$$f = x + yz + \overline{y}\,\overline{z}$$

è espressa mediante gli operatori AND, OR e NOT.
Se effettuiamo una doppia negazione, tenedo conto del teorema di De Morgan, la funzione si può scrivere nella forma:

$$f = \overline{\overline{f}} = \overline{\overline{x} \cdot \overline{yz} \cdot \overline{\overline{y}\,\overline{z}}}.$$

Per eliminare invece l'operatore AND basta applicare ancora il teorema di De Morgan all'espressione ricavata ottenendo:

$$f = \overline{\overline{f}} = \overline{\overline{x} \cdot \overline{yz} \cdot \overline{\overline{y}\,\overline{z}}} = x + \overline{\overline{yz}} + \overline{y + z} = x + \overline{\overline{y} + \overline{z}} + \overline{y + z}.$$

4.2. L'operatore NAND

La negazione del prodotto di due o più variabili si indica mediante il simbolo /:

$$x/y = \overline{xy} = \overline{x} + \overline{y}.$$

Il simbolo / esprime l'operatore NAND (da NOTAND). Il NAND è un operatore universale in quanto realizza, da solo, la negazione e la somma:

$x/x = \overline{x}$	(realizzazione della negazione)
$(x/y)/(x/y) = \overline{\overline{xy}} = xy$	(realizzazione dell'AND)
$(x/x)/(y/y) = \overline{x}/\overline{y} = x + y$	(realizzazione dell'OR).

Le più importanti proprietà dell'operatore NAND sono:

$x/y = y/x$	(proprietà commutativa)
$x/0 = 1$	
$x/1 = \overline{x}$	(altra realizzazione della negazione)
$x/\overline{x} = 1$	
$x/(y/x) \neq (x/y)/z$	(l'operatore non è associativo).

Le funzioni espresse con l'operatore NAND possono essere facilmente trasformate in somma di prodotti, e riportate su una tabella di verità o su una mappa di Karnaugh.

Esempio 18. Trasformare la funzione:

$$f = ((x_1/1)/x_2/x_3)/(x_4/(x_5/1))/(x_6/1)$$

in somma di prodotti.
Eliminando le parentesi partendo dalle più interne si ha:

$$f = (\overline{x}_1/x_2/x_3)/(x_4/\overline{x}_5)/\overline{x}_6 = \overline{\overline{x}_1/x_2/x_3} + \overline{x_4/\overline{x}_5} + x_6 =$$
$$= \overline{x_1 + \overline{x}_2 + \overline{x}_3} + \overline{\overline{x}_4 + x_5} + x_6 = \overline{x}_1 x_2 x_3 + x_4 \overline{x}_5 + x_6.$$

La regola per passare dalle espressioni contenenti gli operatori / a quelle equivalenti del tipo AND, OR, NOT si deduce dalle proprietà dell'operatore NAND e dal teorema di De Morgan:

1) si eliminano le espressioni del tipo $x/1$, sostituendole con $\overline{x}$;
2) si sostituiscono gli operatori / tra le parentesi con gli operatori +;
3) si eliminano le parentesi sostituendo gli operatori / con gli operatori AND e si invertono tutte le variabili che compaiono da sole in una parentesi.

Esercizio 10. Rappresenta sulla mappa di Karnaugh la funzione:

$$y = (\overline{x}_1/\overline{x}_3)/(\overline{x}_2/x_3)/(x_1/x_2/\overline{x}_3/x_4)/(\overline{x}_1/x_2/x_3/x_4).$$

Dalle proprietà dell'operatore NAND, e dall'osservazione fatta nella sezione 4 si deduce, infine, il procedimento inverso, quello per esprimere una funzione, data come somma di prodotti, con il solo operatore NAND:

1) si sostituisce ogni operatore OR e AND con l'operatore NAND, mettendo tra parentesi tutti i termini della somma;
2) si invertono le variabili che compaiono da sole in una parentesi;
3) si sostituisce, ad ogni variabile negata l'espressione $(x/1)$.

Esempio 19. Esprimere la funzione:

$$f = x_1\overline{x}_2 + x_3x_4 + x_2\overline{x}_5 + x_6$$

con il solo operatore NAND.
Seguendo i tre passi del procedimento indicato si ha:

$$x_1\overline{x}_2 + x_3x_4 + x_2\overline{x}_5 + x_6 = (x_1/\overline{x}_2)/(x_2/\overline{x}_5)/(x_3/x_4)/(\overline{x}_6) =$$
$$= (x_1/(x_2/1))/(x_3/x_4)/(x_2/(x_5/1))/(x_6/1).$$

Domanda 9. Come si esprime la funzione $Y = \overline{A}\,\overline{B} + \overline{A}D + \overline{B}CD + B\overline{C}D$ usando il solo operatore NAND?

4.3. L'operatore NOR

La negazione della somma di due o più variabili si indica mediante il simbolo $\downarrow$:

$$x \downarrow y = \overline{x + y} = \overline{x}\,\overline{y}.$$

Il simbolo $\downarrow$ esprime l'operatore NOR (da NOT–OR). Anche il NOR, essendo il duale del NAND, è un operatore universale in quanto realizza, da solo, la negazione, il prodotto e la somma:

$$x \downarrow x = \overline{x}$$

$$(x \downarrow x) \downarrow (y \downarrow y) = xy$$

$$(x \downarrow y) \downarrow (x \downarrow y) = x + y.$$

Le più importanti proprietà dell'operatore NOR sono:

$x \downarrow y = y \downarrow x$ (proprietà commutativa)

$x \downarrow 1 = 0$

$x \downarrow 0 = \overline{x}$ (altra realizzazione della negazione)

$x \downarrow \overline{x} = 0$

$x \downarrow y \downarrow z \neq (x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$ (l'operatore non è associativo).

Le funzioni espresse con l'operatore NOR possono essere facilmente trasformate in prodotti di somme, e riportate su una tabella di verità o su una mappa di Karnaugh in modo del tutto analogo a quanto visto per l'operatore NAND.
La regola per passare dalle espressioni contenenti gli operatori a quelle equivalenti del tipo NOT–AND–OR si deduce dalle proprietà dell'operatore NOR e dal teorema di De Morgan:

1) si eliminano le espressioni del tipo $x \downarrow 0$, sostituendole $\overline{x}$;
2) si sostituiscono gli operatori tra le parentesi con gli operatori AND e OR;
3) si sostituiscono tutti gli operatori nelle parentesi con gli operatori AND e OR, e si invertono tutte le variabili che compaiono da sole in una parentesi.

Dalle proprietà dell'operatore NOR, e dall'osservazione fatta nella sezione 4, si deduce infine il procedimento inverso, quello per esprimere una funzione, data come prodotto di somme, con il solo operatore NOR:

1) si sostituisce ogni operatore AND e OR con l'operatore $\downarrow$, mettendo tra parentesi tutti i termini del prodotto;
2) si invertono le variabili che compaiono da sole in una parentesi;
3) si sostituisce, ad ogni variabile negata l'espressione $(x \downarrow 0)$.

Esempio 20. Esprimere la funzione:

$$f = (x_1 + \overline{x}_2)(x_3 + x_4)(x_2 + x_5)x_6$$

con il solo operatore NOR.
Seguendo i tre passi del procedimento indicato si ha:

$$f = (x_1 \downarrow \overline{x}_2) \downarrow (x_3 \downarrow x_4) \downarrow (x_2 \downarrow x_5) \downarrow x_6 =$$
$$= (x_1 \downarrow (x_2 \downarrow 0) \downarrow (x_3 \downarrow x_4) \downarrow (x_2 \downarrow x_5) \downarrow (x_6 \downarrow 0)).$$

Domanda 10. Come si esprime la funzione:

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

usando il solo operatore NOR?

In questa sezione abbiamo esaminato, con un certo dettaglio, proprietà ed usi degli operatori universali NAND e NOR. In particolare abbiamo imparato a:

- esprimere una funzione di commutazione usando il solo operatore NAND;
- esprimere una funzione di commutazione usando il solo operatore NOR;
- esprimere una funzione contenente l'operatore NAND nella forma "somma di prodotti";
- esprimere una funzione contenente l'operatore NOR nella forma "prodotto di somme".

Gli operatori NAND e NOR sono molto importanti, sia dal punto di vista teorico che da quello pratico. Teoricamente, infatti, gli operatori universali sono in grado di esprimere da soli tutte le funzioni logiche; in pratica, poi, è possibile realizzare qualsiasi circuito di commutazione usando soltanto elementi che realizzano materialmente questi due operatori.
Osservazione importante: anche se NAND e NOR sono due operatori universali, spesso compaiono nelle espressioni logiche anche insieme all'operatore NOT. Questo semplifica, allo stesso tempo, la scrittura delle espressioni e la costituzione dei relativi circuiti, come avremo occasione di vedere nelle prossime unità.

5. L'OPERATORE OR ESCLUSIVO

Un operatore logico molto importante è l'OR esclusivo (XOR), detto anche "somma modulo 2" o "anticoincidenza", indicato col simbolo $\oplus$ tra le variabili x e y.
Per due variabili, la tabella di verità della funzione OR esclusivo è:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Se ne deduce l'espressione analitica:

$$x \oplus y = x\overline{y} + \overline{x}y.$$

La funzione si chiama OR esclusivo perché vale 1 quando l'una o l'altra, ma non entrambe, le sue variabili valgono 1; si chiama anche anticoincidenza perché vale 1 quando le due variabili sono diverse.
Le più importanti proprietà dell'operatore sono:

$$x \oplus y = y \oplus x \qquad \text{(proprietà commutativa)}$$
$$x \oplus (y \oplus z) = (x \oplus y) \oplus z \qquad \text{(proprietà associativa)}$$
$$x \oplus 1 = \overline{x}$$
$$x \oplus 0 = x$$
$$x \oplus x = 0$$
$$x \oplus \overline{x} = 1.$$

L'operatore non è universale, in quanto non è in grado di realizzare, da solo, le operazioni di somma e prodotto.
L'operatore $\oplus$ applicato a n variabili definisce la funzione di disparità:

$$P = x_1 \oplus x_2 \oplus \cdots \oplus x_n.$$

La funzione P è chiamata di disparità perché vale 1 se e solo se un numero dispari di variabili vale 1.
Le espressioni contenenti l'OR esclusivo su più di due variabili possono essere trasformate in somme di prodotti sfruttando la proprietà associativa. Osserva l'esempio seguente.

Esempio 21. La funzione $f = x \oplus y \oplus z$ può essere trasformata in:

$$x \oplus y \oplus z = (x \oplus y) \oplus z = (\overline{x}y + x\overline{y}) \oplus z = \overline{(\overline{x}y + x\overline{y})}z + (\overline{x}y + x\overline{y})\overline{z} =$$
$$= (x + \overline{y})(\overline{x} + y)z + \overline{x}y\overline{z} + x\overline{y}\,\overline{z} = xyz + \overline{x}\,\overline{y}\,z + xy\overline{z} + \overline{x}y\overline{z} + x\overline{y}\,\overline{z}.$$

Nota che la forma finale della f poteva essere ottenuta direttamente come somma di tutti i mintermini corrispondenti a configurazioni che contengono un numero dispari di 1.

Esercizio 11. Scrivi la forma minima come somma di prodotti della funzione di quattro variabili:

$$y = x_1 \oplus x_2 \oplus x_3 \oplus x_4.$$

Osservazione importante: la negazione della funzione $x \oplus y$, cioè $\overline{x \oplus y}$, controlla la coincidenza delle due variabili x e y in quanto vale 1 quando $x = y$.
La funzione $\overline{x \oplus y}$ viene spesso indicata con $x \equiv y$. L'operatore $\equiv$ si chiama operatore coincidenza, o equivalenza, o "XNOR".
Abbiamo visto diversi operatori, e abbiamo imparato a maneggiarli con una certa disinvoltura. Per dare un ultimo sguardo a tutti, li riassumiamo in una tabella riassuntiva.

Operatore	Simbolo	Definizione
NOT	$y = \overline{x}$	$y = 1$ quando $x = 0$
AND	$x \cdot y$	$f = 1$ quando x e y sono entrambi 1
OR	$f = x + y$	$f = 0$ quando x e y sono entrambi 0
NAND	$f = x/y$	$f = 0$ quando x e y sono entrambi 1
NOR	$f = x \downarrow y$	$f = 1$ quando x e y sono entrambi 0
XOR	$f = x \oplus y$	$f = 1$ quando x è diverso da y
XNOR	$f = x \equiv y$	$f = 1$ quando $x = y$

Esercizio 12. Realizzare una funzione di sei variabili che vale 1 quando un numero dispari delle prime tre variabili e un numero pari delle altre tre variabili vale 1.

6. PORTE LOGICHE E LORO SIMBOLI GRAFICI

6.1. Introduzione

Si è accennato molte volte all'esistenza di componenti fisici capaci di presentare due stati diversi ai quali, per convenzione, si fanno corrispondere i simboli 0 e 1 dell'algebra booleana o di un codice binario, oppure le cifre 0 e 1 del sistema di numerazione binario.
Nei circuiti logici (detti anche circuiti di commutazione o circuiti digitali) di tipo elettronico i valori 0 e 1 sono assegnati a due valori ben definiti della tensione elettrica: il valore "alto" (H) e il valore "basso" (L). Le due lettere, iniziali delle parole inglesi high (alto) e low (basso) vanno intese come tensione più positiva (H) e tensione meno positiva (L); l'effettiva corrispondenza tra i simboli 0, 1 e le due tensioni H, L è arbitrario. Se si adotta la corrispondenza:

$$1 \rightarrow H \quad \text{e} \quad 0 \rightarrow L$$

si dice che si lavora in logica positiva, mentre nel caso opposto si dice che si lavora in logica negativa.
Si definisce porta logica un circuito elettronico a n ingressi ed una uscita in cui i segnali di ingresso e di uscita assumono i due stati H e L e il legame tra ingressi e uscita è definito da uno degli operatori dell'algebra booleana. Nel seguito faremo riferimento alla logica positiva, facendoti subito osservare che per il principio di dualità se si adotta la logica negativa le porte logiche si comportano in modo duale e precisamente:

- AND diventa OR;
- NAND diventa NOR;
- XOR diventa XNOR

  e viceversa.

Le porte logiche permettono di realizzare qualsiasi circuito logico allo stesso modo in cui i corrispondenti operatori dell'algebra boolana realizzano qualsiasi funzione di commutazione.

6.2. Simboli grafici

Indipendentemente dalla loro realizzazione circuitale, le porte logiche sono indicate sempre con lo stesso simbolo. Nella figura 17 sono riportati i simboli grafici più comunemente usati per rappresentare le porte logiche corrispondento agli operatori che abbiamo visto nelle sezioni precedenti.

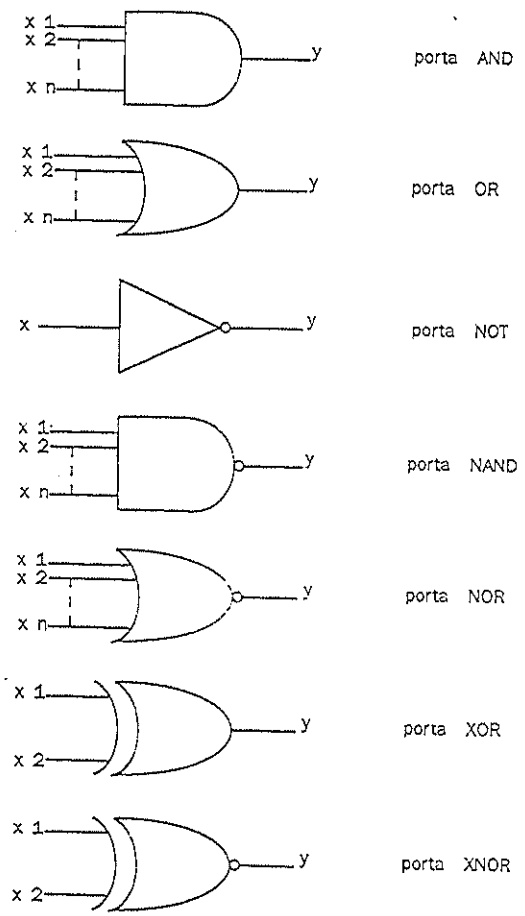


Figura 17. *Simboli delle porte logiche.*

Si noti che il cerchietto all'uscita delle porte indica negazione per cui è giustificato che la porta NAND che può essere vista come un AND in serie ad un NOT è indicato col simbolo dell'AND con un cerchietto sull'uscita.

Molte volte troverai circuiti logici che hanno ingressi col cerchietto: ciò sta ad indicare che il circuito effettua una negazione delle variabili corrispondenti.
Ricordando il teorema di De Morgan è facilmente riconducibile un circuito di commutazione di tutti NAND o di tutti NOR ad un circuito OR di AND o AND di OR come risulta evidente dalla figura 18 nel caso di un circuito NAND di NAND.
Nel caso di circuito NOR di NOR si procede nello stesso modo arrivando ad un circuito AND di OR.

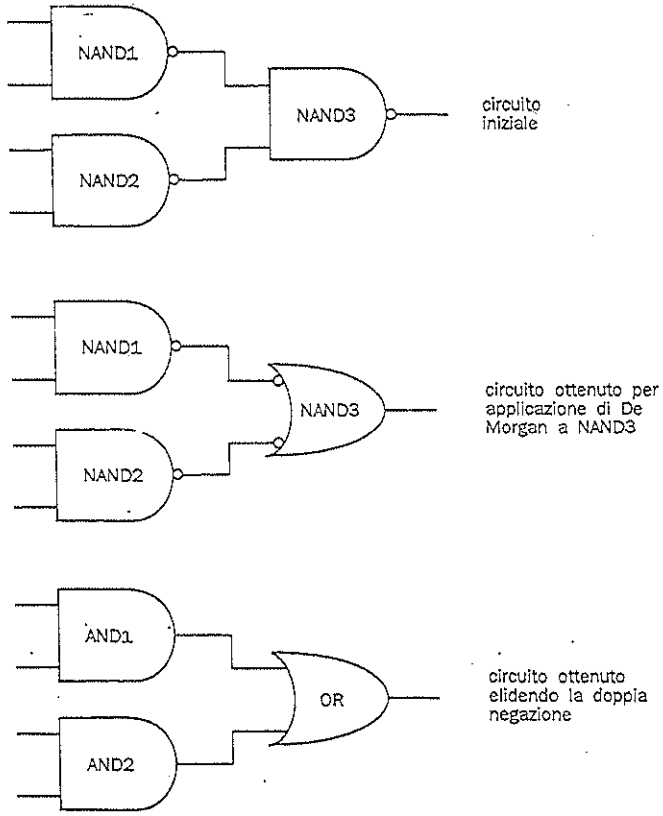


Figura 18. *Trasformazione di un circuito NAND–NAND in un circuito OR di AND.*

7. CONCLUSIONI

7.1. Sommario

Questa unità, che come la precedente, ha carattere propedeutico, è dedicata all'algebra booleana, o algebra di commutazione. Abbiamo trattato questo argomento procedendo per definizioni successive, e cercando di semplificarlo al massimo. Notiamo in particolare che l'algebra di commutazione è fondata su un insieme di due elementi, indicati convenzionalmente con i simboli 0 e 1. Questi simboli non hanno nulla a che vedere con i simboli che abbiamo usato nel sistema di numerazione binarie nella teoria dei codici, tuttavia è possibile effettuare operazioni logiche su stringhe di bit che possono essere numeri binari o parole di codici binari nel modo che vedremo nel seguito del corso.
Riassumendo brevemente quanto abbiamo introdotto in questa unità, ricordiamo che, dopo aver introdotto l'algebra di commutazione, ci siamo soffermati particolarmente sul concetto di *funzione*. Le funzioni di commutazione sono un argomento di fondamentale importanza, perché permettono, tra l'altro, di descrivere con metodi algebrici il funzionamento di qualsiasi circuito. Abbiamo visto che le funzioni possono essere rappresentate sotto forma di tabelle (*tabelle di verità*) e sotto diverse forme analitiche (*forme canoniche* e *forme semplificate*). La possibilità di avere forme semplificate di una funzione, cui corrispondono circuiti più semplici, ci ha portato a considerare il problema della semplificazione delle funzioni. Tale problema può essere risolto in modo algebrico, sfruttando i teoremi dell'algebra booleana, o più efficacemente con un metodo grafico (*mappe di Karnaugh*), che però può essere applicato solo per un numero limitato di variabili. Ci siamo serviti di questo metodo anche per introdurre e minimizzare le funzioni parzialmente specificate. Abbiamo infine introdotto degli operatori fondamentali per la sintesi dei circuiti: *NAND*, *NOR* e *XOR*. Nell'ultima sezione abbiamo introdotto le porte logiche e i simboli con cui vengono rappresentate negli schemi circuitali e abbiamo visto come è facilmente trasformabile un circuito di soli NAND o di soli NOR in un circuito equivalente costituito da OR di AND e AND di OR rispettivamente.

7.2. Risposte alle domande

Domanda 1. 1.

Domanda 2. $\overline{x+y+z+w} = \overline{x}\,\overline{y}\,\overline{z}\,\overline{w}$.

Domanda 3. La tabella è la seguente:

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

Domanda 4. La tabella è la seguente:

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

Domanda 5. La funzione è:

$$\overline{x}z + \overline{y}z + xy\overline{z}.$$

Domanda 6. La forma equivalente è stata ottenuta scegliendo gli implicanti segnati nella figura seguente:

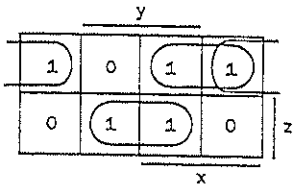


Figura 19.

Domanda 7. Poiché la funzione è apparentemente di quattro variabili essa è rappresentata dalla seguente mappa di tre variabili:

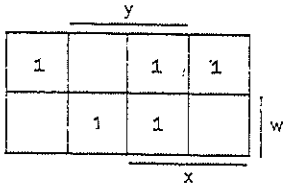


Figura 20.

Domanda 8. Sulla mappa di Karnaugh, la funzione si rappresenta nel modo indicato nella seguente figura.

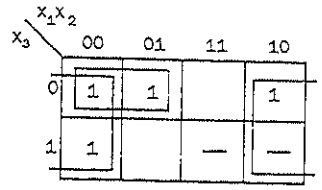


Figura 21.

La forma minima è: $f = \overline{x}_1\overline{x}_3 + \overline{x}_2$.

Domanda 9. $y = (\overline{A}/\overline{B})/(\overline{A}/D)/(\overline{B}/C/D)/(B/\overline{C}/D) =$
$= ((A/1)/(B/1))/((A/1)/D)/((B/1)/C/D)/(B/(C/1)/D).$

Domanda 10. $(\overline{A} \downarrow B \downarrow \overline{C}) \downarrow (D \downarrow \overline{E}) \downarrow F$

$(\overline{A} \downarrow B \downarrow \overline{C}) \downarrow (D \downarrow \overline{E}) \downarrow (\overline{F})$

$y = ((A \downarrow 0) \downarrow B \downarrow (C \downarrow 0)) \downarrow (D \downarrow (E \downarrow 0)) \downarrow (F \downarrow 0).$

7.3. Soluzioni degli esercizi

Esercizio 1. Le funzioni di due variabili sono $16 (2^{2^2})$, e sono riportate nella seguente tabella:

x_1	x_2	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
0	0	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0

x_1	x_2	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	y_{16}
0	0	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1

y_1 è l'identità $y = 0$; y_{16} è l'identità $y = 1$.

Esercizio 2. La tabella è la seguente:

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

Esercizio 3. La funzione ha la tabella di verità simile a quella nella domanda 3 e pertanto la sua forma canonica in prodotto di somme si ottiene considerando gli "zeri" della funzione:

$$f(a,b) = (a+b)(\overline{a}+\overline{b}).$$

Esercizio 4. $y = \sum(1,2,4) \quad y = \prod(0,3,5,6,7).$

Esercizio 5. La tabella di verità della y è:

X	x_1	x_2	x_3	y
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

per cui:

$$y = \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2x_3 + x_1x_2x_3 =$$
$$= \overline{x}_1\overline{x}_2(\overline{x}_3 + x_3) + (\overline{x}_1 + x_1)x_2x_3$$

pertanto:

$$y = \overline{x}_1\overline{x}_2 + x_2x_3.$$

Esercizio 6. La mappa è la seguente:

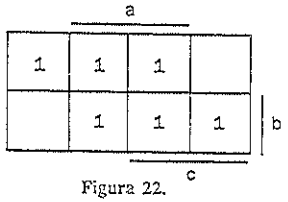


Figura 22.

Esercizio 7. La mappa è riportata nella seguente figura:

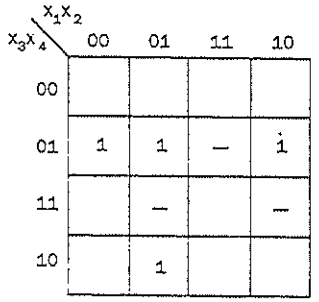


Figura 23.

Esercizio 8. La funzione è rappresentata nella mappa di figura 24; i blocchi formati permettono di scrivere la funzione nella forma:

$$y = x_1\overline{x}_2 + x_3x_4x_5 + \overline{x}_1x_2\overline{x}_3\overline{x}_4x_5.$$

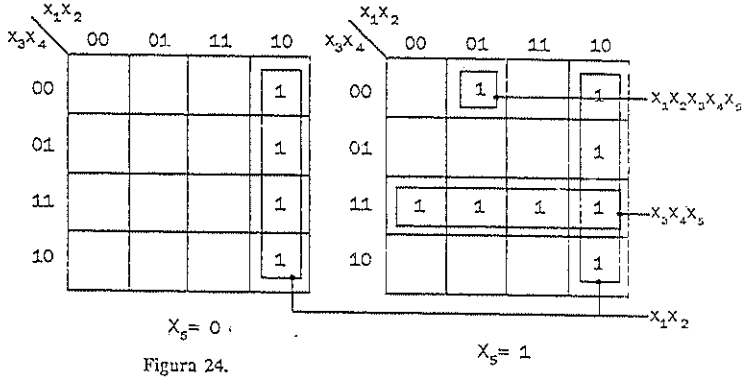


Figura 24.

Esercizio 9. Nella figura 25 è rappresentata la mappa di Karnaugh per la funzione:

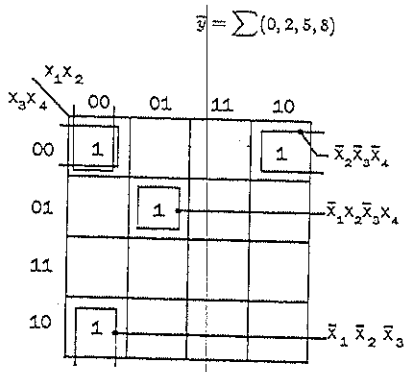


Figura 25.

pertanto:

$$y = \overline{\overline{y}} = \overline{\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3x_4 + \overline{x}_1\overline{x}_2\overline{x}_4} =$$
$$= (x_2 + x_3 + x_4)(x_1 + \overline{x}_2 + x_3 + \overline{x}_4)(x_1 + x_2 + x_4).$$

Esercizio 10. La funzione, espressa come somma di prodotti, è:

$$y = (\overline{x}_1/\overline{x}_3)/(\overline{x}_2/x_3)/(x_1/x_2/\overline{x}_3/x_4)/(\overline{x}_1/x_2/x_3/x_4) =$$
$$= \overline{(\overline{x}_1/\overline{x}_3)(\overline{x}_2/x_3)(x_1/x_2/\overline{x}_3/x_4)(\overline{x}_1/x_2/x_3/x_4)} =$$
$$= \overline{\overline{x}_1/\overline{x}_3} + \overline{\overline{x}_2/x_3} + \overline{x_1/x_2/\overline{x}_3/x_4} + \overline{\overline{x}_1/x_2/x_3/x_4} =$$
$$= \overline{x_1 + x_3} + \overline{x_2 + \overline{x}_3} + \overline{\overline{x}_1 + \overline{x}_2 + x_3 + \overline{x}_4} + \overline{x_1 + \overline{x}_2 + \overline{x}_3 + \overline{x}_4} =$$
$$= \overline{x}_1\overline{x}_3 + \overline{x}_2x_3 + x_1x_2\overline{x}_3x_4 + \overline{x}_1x_2x_3x_4.$$

La mappa di Karnaugh della funzione y è riportata nella figura 26.

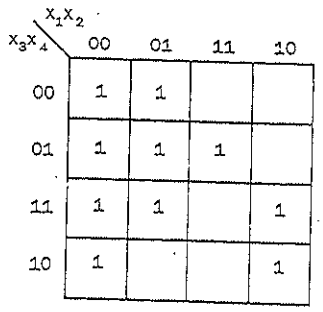


Figura 26.

Esercizio 11. $y = \overline{x}_1x_2x_3x_4 + x_1\overline{x}_2x_3x_4 + x_1x_2\overline{x}_3x_4 + x_1x_2x_3\overline{x}_4 +$
$+ x_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2x_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_3x_4$
(la funzione non è semplificabile).

Esercizio 12. $f = (x_1 \oplus x_2 \oplus x_3) \cdot \overline{x_4 \oplus x_5 \oplus x_6}$.