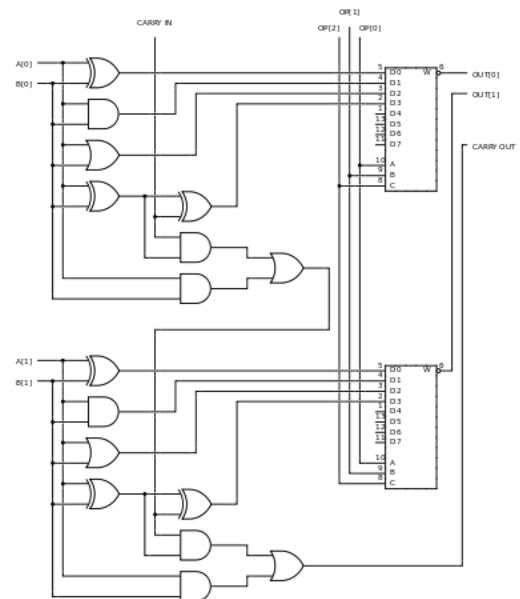
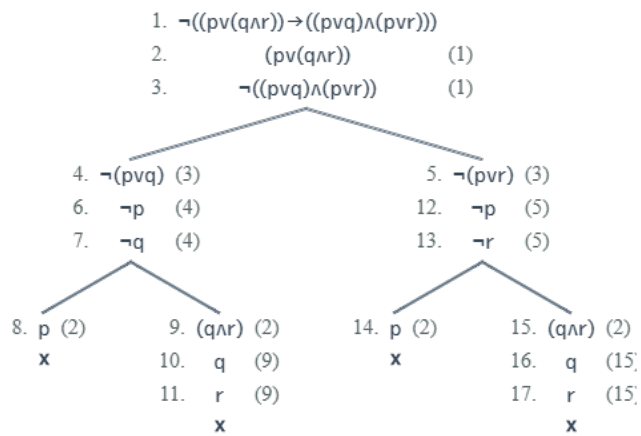


Dispense di Logica e reti logiche

Emanuele Izzo, 0253052



Corso di laurea triennale Informatica
 Università Tor Vergata, Facoltà di Scienze MM.FF.NN.
 11/11/2020

Documento realizzato in L^AT_EX

Indice

I	Logica	4
1	Le dimostrazioni per assurdo e per induzione	5
1.1	Dimostrazione per assurdo	5
1.2	Dimostrazione per induzione	5
2	L'ipercubo d-dimensionale	7
2.1	Distanza di Hamming	7
3	La logica proposizionale	8
3.1	Struttura delle formule	8
3.1.1	I simboli	8
3.1.2	La sintassi	8
3.1.3	La semantica	9
3.2	CNF e DNF	9
3.2.1	Forma normale congiuntiva (CNF)	9
3.2.2	Forma normale disgiuntiva (DNF)	9
3.3	Il metodo dei tableaux	9
3.4	Conseguenza logica	10
3.5	Correttezza e completezza del metodo dei tableaux	10
3.6	Funzione soddisfacibile	11
3.7	Proprietà dei tableaux	11
3.8	Lemma di Hiurtikka	11
4	La logica del primo ordine	12
4.1	Struttura delle formule	12
4.1.1	I simboli	12
4.1.2	La sintassi	12
4.1.3	La semantica	12
4.2	Il metodo dei tableaux	13
4.3	Correttezza e completezza del metodo dei tableaux	13
4.4	Formule pure e chiuse	13
4.5	Funzione soddisfacibile	14
4.6	Insieme di Hiurtikka	14
4.7	Lemma di Hiurtikka	14
5	I sistemi assiomatici	15
5.1	La regola d'inferenza <i>modus ponens</i> (MP)	15
5.2	Dimostrazione in un sistema assiomatico	15
5.3	Teorema di deduzione	15

5.4	Derivazione di una formula	15
II	Reti logiche	17
6	Conversione di base	18
6.1	Le basi	18
6.2	Conversione da una base b in base 10	19
6.3	Conversione dalla base 10 ad una base b	19
6.4	Conversioni tra la base 8 e la base 2	19
6.4.1	Convertire dalla base 8 alla base 2	19
6.4.2	Convertire dalla base 2 alla base 8	19
6.5	Conversioni tra la base 16 e la base 2	20
6.5.1	Convertire dalla base 16 alla base 2	20
6.5.2	Convertire dalla base 2 alla base 16	20
7	Rappresentazione dei numeri interi in binario	21
7.1	Complemento ad 1 e complemento a 2	22
7.2	Il problema della moltiplicazione in binario	22
7.2.1	L'algoritmo di Karatsuba	23
8	Rappresentazione dei numeri frazionari in binario	24
8.1	Codifica in virgola fissa	24
8.2	Codifica in virgola mobile	24
8.2.1	Codifica a 32 bit	25
8.2.2	Codifica a 64 bit	25
9	L'algebra booleana	26
9.1	Struttura delle formule	26
9.1.1	I simboli	26
9.1.2	La sintassi	26
9.1.3	I postulati	27
9.2	I teoremi ad una variabile	27
9.3	I teoremi a più variabili	27
9.4	Il teorema di De Morgan	27
9.5	Il codice di Gray	28
9.6	La mappa di Karnaugh	28
10	Le porte logiche	30
10.1	Equivalenza di porte (con sole porte NAND e NOR)	31
11	I blocchi funzionali	33
11.1	Decoder	33
11.2	Multiplexer	33
11.3	Half adder	34
11.4	Full adder	34
11.5	Sommatore a propagazione di riporto ad onda	35
11.6	Sommatore/sottrattore	36
11.7	Sommatore ad anticipazione di riporto	36

11.8	Lo shifter	37
12	I circuiti sequenziali	38
12.1	Latch SR	38
12.2	Latch D	38
12.3	Flip-flop D	39
12.4	Registro ad n bit	39
12.5	Circuiti sequenziali	39
13	Le macchine a stati finiti	41
13.1	Macchine alla Moore e macchine alla Mealy	41
13.2	Fattorizzazione	42

Parte I

Logica

Capitolo 1

Le dimostrazioni per assurdo e per induzione

1.1 Dimostrazione per assurdo

La dimostrazione per assurdo viene usata per dimostrare la validità di un teorema, e si basa sul dimostrare che tale teorema è vero se la sua negazione è falsa (raggiungendo un paradosso)

Esempio: teorema di Cantor

Il secondo teorema di Cantor afferma che $|\mathcal{P}(\mathbb{N})| > |\mathbb{N}|$. Possiamo usare la dimostrazione per assurdo, provando che $|\mathcal{P}(\mathbb{N})| = |\mathbb{N}|$ è falso. Poniamo $S_{\mathbb{N}} := \{A_i | A_i \subseteq \mathbb{N}\}_{1 \leq i \leq n}$ l'insieme dei sottinsiemi di \mathbb{N} , dove $n \in \mathbb{N} \setminus \{0\}$. Definiamo $R := \{k \in \mathbb{N} | k \notin A_i\}$ e cerchiamo un $R \subseteq \mathbb{N}$. Poniamo $R = A_i$: in questo modo abbiamo che:

- Se $i \in R \Rightarrow i \in A_i \Rightarrow i \notin R$;
- Se $i \notin R \Rightarrow i \notin A_i \Rightarrow i \in R$.

Ciò ha portato ad un paradosso: pertanto, $|\mathcal{P}(\mathbb{N})| = |\mathbb{N}|$ è falso. Da qui abbiamo che il teorema di Cantor è vero.

1.2 Dimostrazione per induzione

La dimostrazione per induzione viene usata per dimostrare che $\forall n_0 > n, P(n)$ è vera. La dimostrazione si svolge in due punti:

- Passo base: dimostrare $P(n_0)$;
- Passo induttivo: ponendo che $P(n)$ è vera, dimostrare che $P(n+1)$ è vera.

Esempio: somma dei primi n numeri dispari

Vogliamo dimostrare che $\forall n > 1, \sum_{k=1}^n (2k-1) = n^2$. Possiamo usare la dimostrazione per induzione, ponendo $P(n) \rightarrow \sum_{k=1}^n (2k-1) = n^2$

- Passo base: $P(1) \rightarrow \sum_{k=1}^1 (2k-1) = 1^2 \rightarrow 1 = 1$;
- Passo induttivo: $P(n+1) \rightarrow \sum_{k=1}^{n+1} (2k-1) = (2(n+1)-1) + \sum_{k=1}^n (2k-1)$. Ponendo
che $\sum_{k=1}^n (2k-1) = n^2$ è vera, abbiamo
 $(2(n+1)-1) + n^2 = n^2 + 2n + 1 = (n+1)^2$.

Capitolo 2

L'ipercubo d -dimensionale

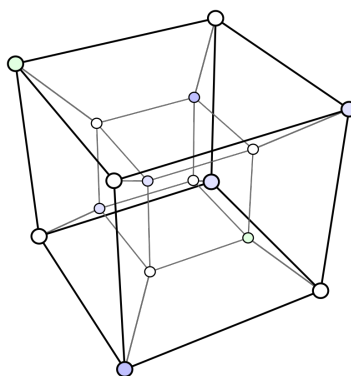


Figura 2.1: Esempio di ipercubo 4-dimensionale

Sia $d \in \mathbb{N} \setminus \{0\}$, si definisce ipercubo d -dimensionale un grafo $G := (V, E)$ avente

- $V := \{\underline{x} := (x_1, x_2, \dots, x_d) \in \{0, 1\}^d\};$
- $G := \{(\underline{x}, \underline{y}) \in V \times V \mid H(\underline{x}, \underline{y}) = 1\}$

2.1 Distanza di Hamming

Viene definita distanza di Hamming la funzione $H(\underline{x}, \underline{y}) = \sum_{i=1}^d |x_i - y_i|$, dove $d \in \mathbb{N} \setminus \{0\}$ e $\underline{x}, \underline{y} \in V$.

Capitolo 3

La logica proposizionale

3.1 Struttura delle formule

3.1.1 I simboli

La logica proposizionale usa tre tipologie di simboli:

- Variabili: p, q, r, \dots oppure p_1, p_2, \dots ;
- Costanti:
 - Vero: $1, t$ [true];
 - Falso: $0, f$ [false];
- Connettivi:
 - Negazione: \sim, \neg [not];
 - Congiunzione logica: \wedge [and];
 - Disgiunzione logica: \vee [or];
 - Implicazione: \Rightarrow ;
 - Negazione congiuntiva: \downarrow [nor];
 - Negazione alternativa: $|$ [nand];
 - Equivalenza: \equiv .

3.1.2 La sintassi

La sintassi delle formule della logica proposizionale è la seguente:

- Variabili e costanti sono formule;
- Se X è una formula, allora anche $\sim X$ è una formula;
- Se X e Y sono formule, allora anche $X \wedge Y, X \vee Y, X \Rightarrow Y, X \downarrow Y, X|Y, X \equiv Y$ sono formule;
- Nient'altro è una formula.

3.1.3 La semantica

Le formule della logica proposizionale possono avere 4 diversi significati:

- Tautologia: la formula è sempre vera;
- Contraddizione: la formula è sempre falsa;
- Contingenza: la formula è vera o falsa in base alla sua interpretazione;

Con interpretazione si intende l'assegnazione di valori di verità alle variabili.

3.2 CNF e DNF

3.2.1 Forma normale congiuntiva (CNF)

Una formula si dice in forma normale congiuntiva (CNF) se è una congiunzione di clausole disgiuntive (dette anche clausole) $f = d_1 \wedge d_2 \wedge \dots \wedge d_n$, dove ogni clausola è una disgiunzione di letterali $d_i = l_{i,1} \vee l_{i,2} \vee \dots \vee l_{i,m}$, e ogni letterale è uguale ad una variabile o alla sua negata.

3.2.2 Forma normale disgiuntiva (DNF)

Una formula si dice in forma normale disgiuntiva (DNF) se è una disgiunzione di clausole congiuntive (dette anche clausole) $f = c_1 \vee c_2 \vee \dots \vee c_n$, dove ogni clausola è una congiunzione di letterali $c_i = l_{i,1} \wedge l_{i,2} \wedge \dots \wedge l_{i,m}$, e ogni letterale è uguale ad una variabile o alla sua negata.

3.3 Il metodo dei tableaux

Il metodo dei tableaux è un metodo di dimostrazione (del tipo "refutation system", simile alla dimostrazione per assurdo) che si basa sullo scomporre la formula opposta di una data e dimostrarne che non esista alcuna combinazione di valori che rende vera la formula opposta (definendo quindi quella data come tautologia). Le formule si dividono in due gruppi (si esclude l'insieme η):

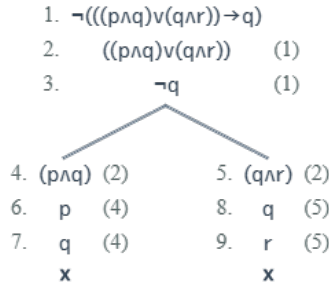


Figura 3.1: Esempio di tableaux (le formule α si splittano verticalmente [righe 1 \rightarrow 2, 3], mentre quelle β orizzontalmente [righe 2 \rightarrow 4, 5])

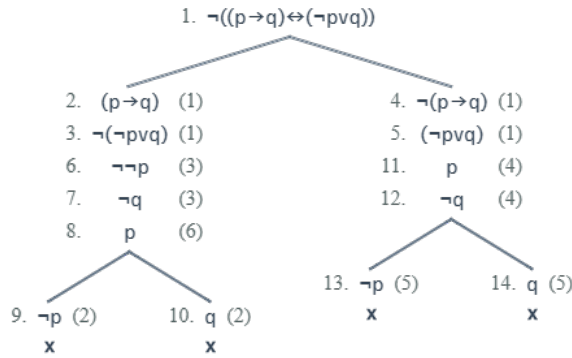


Figura 3.2: Esempio di tableaux (le formule η si splittano sia verticalmente che orizzontalmente [righe 1 \rightarrow [2, 3], [4, 5]])

- Formule di tipo α (vere in base a tutte e due le variabili):

α	α_1	α_2
$x \wedge y$	x	y
$\sim (x \vee y)$	$\sim x$	$\sim y$
$\sim (x \Rightarrow y)$	x	$\sim y$

- Formule di tipo β (vere in base a una o all'altra variabile):

β	β_1	β_2
$x \vee y$	x	y
$\sim (x \wedge y)$	$\sim x$	$\sim y$
$x \Rightarrow y$	$\sim x$	y

- Formule di tipo η (speciali):

η	η_{11} η_{12}	η_{21} η_{22}
$x \equiv y$	x y	$\sim x$ $\sim y$
$\sim (x \equiv y)$	x $\sim y$	$\sim x$ y

3.4 Conseguenza logica

Date le formule X_1, X_2, \dots, X_n , allora Y è la conseguenza logica di X_1, X_2, \dots, X_n se $X_1 \wedge X_2 \wedge \dots \wedge X_n \Rightarrow Y$ è una tautologia (da cui sappiamo che, ponendo $X = X_1 \wedge X_2 \wedge \dots \wedge X_n$, e sapendo che $X \Rightarrow Y$ è vero, allora X e Y sono veri, e quindi X_1, X_2, \dots, X_n sono veri).

3.5 Correttezza e completezza del metodo dei tableaux

Se abbiamo una "dimostrazione" tramite tableaux di una formula X , allora X è una tautologia (correttezza). Se X è una tautologia, allora esiste una dimostrazione per X tramite il metodo dei tableaux (completezza).

3.6 Funzione soddisfacibile

Una formula X si dice soddisfacibile se esiste un'interpretazione che la rende vera. Un insieme di formule S è soddisfacibile se esiste un'interpretazione che soddisfa tutte le formule di S . Sia X una formula soddisfacibile:

- Se X è di tipo α , allora $\{X, \alpha_1, \alpha_2\}$ è soddisfacibile;
- Se X è di tipo β , allora $\{X, \beta_1\}$ o $\{X, \beta_2\}$ è soddisfacibile.

3.7 Proprietà dei tableaux

Sia T un tableau "completo", sia Θ un ramo aperto, sia S l'insieme delle formule sul ramo T , allora:

H₁. In S non sono presenti contemporaneamente una variabile e la sua negata;

H₂. Presa una formula α compresa in S , allora S comprende α_1 e α_2 ;

H₃. Presa una formula β compresa in S , allora S comprende o β_1 o β_2 .

3.8 Lemma di Hiurtikka

Un insieme di formule S che soddisfa H₁, H₂ ed H₃ è soddisfacibile.

Capitolo 4

La logica del primo ordine

4.1 Struttura delle formule

4.1.1 I simboli

La logica proposizionale usa tre tipologie di simboli:

- Variabili: x, y, z, \dots oppure x_1, x_2, \dots ;
- Parametri: a, b, c, \dots oppure a_1, a_2, \dots ;
- Lettere predicative: P, Q, R, \dots oppure P_1, P_2, \dots ;
- Tutti i connettivi della logica proposizionale;
- Quantificatori: \forall (per ogni), \exists (esiste).

4.1.2 La sintassi

La sintassi delle formule della logica proposizionale è la seguente:

- $P(x_1, x_2, \dots, x_n)$ è una formula;
- Se P è una formula, allora anche $\sim P$ è una formula;
- Se P e Q sono formule, allora anche $P \wedge Q$, $P \vee Q$, $P \Rightarrow Q$, $P \downarrow Q$, $P|Q$, $P \equiv Q$ sono formule;
- Se P è una formula e x è una sua variabile, allora $\forall xP$ e $\exists xP$ sono formule;
- Nient'altro è una formula.

4.1.3 La semantica

Il dominio è l'intervallo in cui si muovono le variabili, mentre l'interpretazione è l'associazione di "relazioni" alle lettere predicative. Si definisce formula valida una formula che, a prescindere dall'interpretazione è sempre vera, mentre si dice tautologia una formula valida che è l'istanza di una tautologia della logica proposizionale.

4.2 Il metodo dei tableaux

Il metodo dei tableaux è un metodo di dimostrazione (del tipo "refusion system", simile alla dimostrazione per assurdo) che si basa sullo scomporre la formula opposta di una data e dimostrare che non esista alcuna combinazione di valori che rende vera la formula opposta (definendo quindi quella data come tautologia). I tableaux della logica del primo ordine sono un'estensione dei tableaux della logica proposizionale, infatti integrano due ulteriori gruppi di formule:

- Formule di tipo γ (o formule universali): $\forall xP(x)$ e $\sim \exists xP(x)$, che permettono di usare una qualsiasi variabile;
- Formule di tipo δ (o formule esistenziali): $\sim \forall xP(x)$ e $\exists xP(x)$, che obbligano l'uso di una variabile nuova, mai usata;

1.	$\neg \exists y \forall x (Fy \rightarrow Fx)$	
2.	$\neg \forall x (Fa \rightarrow Fx)$	(1)
3.	$\neg (Fa \rightarrow Fb)$	(2)
4.	Fa	(3)
5.	$\neg Fb$	(3)
6.	$\neg \forall x (Fb \rightarrow Fx)$	(1)
7.	$\neg (Fb \rightarrow Fc)$	(6)
8.	Fb	(7)
9.	$\neg Fc$	(7)
	x	

Figura 4.1: Esempio di tableaux (le formule γ permettono di usare una qualsiasi variabile)

1.	$\neg (\forall x Px \rightarrow (\exists x (Px \vee Qx) \wedge \exists x (Px \vee Rx)))$	
2.	$\forall x Px$	(1)
3.	$\neg (\exists x (Px \vee Qx) \wedge \exists x (Px \vee Rx))$	(1)
4.	Pa	(2)
<div style="text-align: center;">└───┬───┘</div>		
5.	$\neg \exists x (Px \vee Qx)$	(3)
6.	$\neg \exists x (Px \vee Rx)$	(3)
7.	$\neg (Pa \vee Qa)$	(5)
8.	$\neg Pa$	(7)
9.	$\neg Qa$	(7)
	x	
10.	$\neg (Pa \vee Ra)$	(6)
11.	$\neg Pa$	(10)
12.	$\neg Ra$	(10)
	x	

Figura 4.2: Esempio di tableaux (le formule δ obbligano l'uso di una variabile nuova, mai usata)

4.3 Correttezza e completezza del metodo dei tableaux

Se abbiamo una "dimostrazione" tramite tableaux di una formula P , allora P è una tautologia (correttezza). Se P è una tautologia, allora esiste una dimostrazione per P tramite il metodo dei tableaux (completezza).

4.4 Formule pure e chiuse

Una formula P si dice pura se non contiene parametri, e si dice chiusa se non contiene variabili libere. Il metodo dei tableaux si applica solo alle formule chiuse.

4.5 Funzione soddisfacibile

Una formula P si dice soddisfacibile se, qualsiasi interpretazione assuma, rimane sempre vera. Sia P una formula soddisfacibile:

- Se P è di tipo α , allora α_1 ed α_2 sono soddisfacibili;
- Se P è di tipo β , allora β_1 o β_2 è soddisfacibile;
- Se P è di tipo γ allora esiste un'interpretazione I che soddisfa P (il valore usato può essere già presente in I oppure no);
- Se P è di tipo δ allora esiste un'interpretazione I che soddisfa P (il valore usato non può essere già presente in I).

4.6 Insieme di Hiutikka

Un insieme di formule S è un insieme di Hiurtikka se:

- H₁. S non contiene sia una formula P che una sua negata;
- H₂. Se S contiene una formula α allora contiene sia α_1 che α_2 ;
- H₃. Se S contiene una formula β allora contiene o β_1 o β_2 ;
- H₄. Se S contiene una formula γ allora contiene $\gamma(a)$ per ogni valore di a ;
- H₅. Se S contiene una formula δ allora contiene $\delta(a)$ per almeno un parametro a .

4.7 Lemma di Hiurtikka

Se S è un insieme di Hiurtikka, allora S è soddisfacibile.

Capitolo 5

I sistemi assiomatici

5.1 La regola d'inferenza *modus ponens* (MP)

Se $X \Rightarrow Y$ è una proposizione vera, e anche la premessa X è vera, allora la conseguenza Y è vera.

$$\frac{X, X \Rightarrow Y}{Y}$$

5.2 Dimostrazione in un sistema assiomatico

Una dimostrazione in sistema assiomatico (\mathcal{S} , MP) è una sequenza di formule, in cui ogni formula:

- O è un'istanza di un'assioma;
- Oppure deriva da MP e da due formule precedenti.

Un teorema in un sistema assiomatico è l'ultima riga di una dimostrazione. Per indicare che X è un teorema in \mathcal{S} si scrive $\vdash_{\mathcal{S}} X$, mentre per indicare che X è una tautologia si scrive $\models X$.

5.3 Teorema di deduzione

Dato Γ un insieme di formule e X, Y due formule, allora $\Gamma, X \vdash Y$ se e solo se $\Gamma \vdash X \Rightarrow Y$.

5.4 Derivazione di una formula

Una derivazione di una formula X a partire da un insieme di formule Γ è una sequenza di formule in ognuna è:

- O un'istanza di un assioma;
- O una formula in Γ ;

- Oppure deriva da MP e da due formule precedenti.

Per indicare che X è derivabile dall'insieme di formule Γ den sistema \mathcal{S} si scrive $\Gamma \vdash_{\mathcal{S}} X$.

Parte II

Reti logiche

Capitolo 6

Conversione di base

6.1 Le basi

In matematica viene usata per la rappresentazione dei numeri la base 10 (l'alfabeto utilizzato è $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$), mentre in informatica vengono principalmente 3 basi:

- *Base 2*: l'alfabeto utilizzato comprende i caratteri $\{0, 1\}$, e un esempio di conversione è il seguente: $80267_{10} = 10011100110001011_2$;
- *Base 8*: l'alfabeto utilizzato comprende i caratteri $\{0, 1, 2, 3, 4, 5, 6, 7\}$, e un esempio di conversione è il seguente: $3407178_{10} = 14776512_8$;
- *Base 16*: indicato di solito con la lettera *H* (*hexadecimal*, esadecimale) l'alfabeto utilizzato comprende i caratteri $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A(10), B(11), C(12), D(13), E(14), F(15)\}$, e un esempio di conversione è il seguente: $988271963_{10} = 3AE7D55B_H$.

Base 10	Base 2	Base 8	Base 16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Tabella 6.1: Conversione dei numero da 0 a 15 in base 2, in base 8 e in base 16 (notare il numero di cifre necessarie per rappresentare i vari)

6.2 Conversione da una base b in base 10

Sia $x_b = d_n d_{n-1} \dots d_1 d_0$ un numero scritto in base b , il suo equivalente scritto in base 10 sarà $x_{10} = d_n b^n + d_{n-1} b^{n-1} + \dots + d_1 b^1 + d_0 b^0$.

6.3 Conversione dalla base 10 ad una base b

Sia x_{10} un numero scritto in base 10, il suo equivalente scritto in base b sarà $x_b = d_n d_{n-1} \dots d_1 d_0$, dove:

$$\begin{cases} x = q_0 * b + d_0 \\ q_0 = q_1 * b + a_1 \\ \dots \\ q_{n-2} = q_{n-1} * b + a_{n-1} \\ q_{n-1} = 0 * b + a_n \end{cases}$$

In generale, questo sistema termina quando il resto $q_i = 0$.

6.4 Conversioni tra la base 8 e la base 2

Base 2	Base 8
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Tabella 6.2: Conversione tra la base 2 e la base 8

6.4.1 Convertire dalla base 8 alla base 2

Per convertire un numero x_8 dalla base 8 alla base 2, è sufficiente convertire ogni cifra del numero x_8 nella relativa sequenza binaria. Esempio:

$$\begin{aligned} 1443314_8 &= \underbrace{1}_{001} \underbrace{4}_{100} \underbrace{4}_{100} \underbrace{3}_{011} \underbrace{1}_{001} \underbrace{4}_{100} \\ 1443314_8 &= 11001000110001100_2 \end{aligned}$$

6.4.2 Convertire dalla base 2 alla base 8

Per convertire un numero x_2 dalla base 2 alla base 8, è sufficiente convertire ogni terna di cifre del numero x_2 nel relativo numero in base 8. Per effettuare questa conversione

è necessario partire dalla cifra meno significativa, e, nel caso siano necessari, si possono aggiungere degli zero dopo la più significativa. Esempio:

$$10110101110011000010_2 = \underbrace{010}_2 \underbrace{110}_6 \underbrace{101}_5 \underbrace{110}_6 \underbrace{011}_3 \underbrace{000}_0 \underbrace{010}_2$$

$$10110101110011000010_2 = 2656302_8$$

6.5 Conversioni tra la base 16 e la base 2

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

Tabella 6.3: Conversione tra la base 2 e la base 16

6.5.1 Convertire dalla base 16 alla base 2

Per convertire un numero x_H dalla base 16 alla base 2, è sufficiente convertire ogni cifra del numero x_H nella relativa sequenza binaria. Esempio:

$$97C5B4_H = \underbrace{9}_{1001} \underbrace{7}_{0111} \underbrace{C}_{1100} \underbrace{5}_{0101} \underbrace{B}_{1011} \underbrace{4}_{0100}$$

$$97C5B4_H = 1001011111000100110110100_2$$

6.5.2 Convertire dalla base 2 alla base 16

Per convertire un numero x_2 dalla base 2 alla base 16, è sufficiente convertire ogni quaterna di cifre del numero x_2 nel relativo numero in base 8. Per effettuare questa conversione è necessario partire dalla cifra meno significativa, e, nel caso siano necessari, si possono aggiungere degli zero dopo la più significativa. Esempio:

$$11100101010101011100000_2 = \underbrace{0111}_7 \underbrace{0010}_2 \underbrace{1010}_A \underbrace{1010}_A \underbrace{1110}_E \underbrace{0000}_0$$

$$11100101010101011100000_2 = 72AAE0_H$$

Capitolo 7

Rappresentazione dei numeri interi in binario

Poniamo di voler rappresentare tutti i possibili numeri interi rappresentabili in binario (base 2) con 4 bit, gli insiemi di numeri rappresentabili variano da tecnica a tecnica di rappresentazione, come mostrato in tabella 7.1. Come possiamo osservare, utilizzando 4 bit:

- La codifica senza segno permette di rappresentare i numeri nell'intervallo $[0, 15] \rightarrow [0, 2^4 - 1]$;
- La codifica con complemento a 2 permette di rappresentare i numeri nell'intervallo $[-8, 7] \rightarrow [-2^3, 2^3 - 1]$;
- La codifica con modulo e segno permette di rappresentare i numeri nell'intervallo $[-7, 7] \rightarrow [-2^3 + 1, 2^3 - 1]$.

Generalizzando, e quindi utilizzando n bit:

- La codifica senza segno permette di rappresentare i numeri nell'intervallo $[0, 2^n - 1]$;
- La codifica con complemento a 2 permette di rappresentare i numeri nell'intervallo $[-2^{n-1}, 2^{n-1} - 1]$;
- La codifica con modulo e segno permette di rappresentare i numeri nell'intervallo $[-2^{n-1} + 1, 2^{n-1} - 1]$.

Base 10	Base 2		
	Senza segno	Complemento a 2	Modulo e segno
-8	—	1000	—
-7	—	1001	1111
-6	—	1010	1110
-5	—	1011	1101
-4	—	1100	1100
-3	—	1101	1010
-2	—	1110	1001
-1	—	1111	1001
0	0000	0000	1000 0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	0101	0101
6	0110	0110	0110
7	0111	0111	0111
8	1000	—	—
9	1001	—	—
10	1010	—	—
11	1011	—	—
12	1100	—	—
13	1101	—	—
14	1110	—	—
15	1111	—	—

Tabella 7.1: Codifiche possibili utilizzando 4 bit

7.1 Complemento ad 1 e complemento a 2

Dato il numero binario $B = (b_n b_{n-1} \dots b_1 b_0)_2$, è possibile definire due complementi per questo numero:

- *Complemento ad 1*: $B = (b_n b_{n-1} \dots b_1 b_0)_2 \rightarrow \bar{B}^{(1)} = (\bar{b}_n \bar{b}_{n-1} \dots \bar{b}_1 \bar{b}_0)_2$, dove $\forall i \in [0, n], \bar{b}_i = |b_i - 1|$ (ciò è possibile dato che b_i può valere solo 0 o 1);
- *Complemento a 2*: $B = (b_n b_{n-1} \dots b_1 b_0)_2 \rightarrow \bar{B}^{(2)} = \bar{B}^{(1)} + 1_2$.

7.2 Il problema della moltiplicazione in binario

Poniamo di voler moltiplicare due numeri in binario, $A = (a_n a_{n-1} \dots a_1 a_0)$ e $B = (b_n b_{n-1} \dots b_1 b_0)$, il prodotto tra i due numeri sarà uguale alla somma dei vari prodotti tra il numero A e i bit del numero B . In altre parole, moltiplicare due numeri in binario corrisponde a sommare i vari A_{b_i} , dove A_{b_i} è uguale ad A shiftato a sinistra di i (questo solo se $b_i = 1$).

7.2.1 L'algoritmo di Karatsuba

L'algoritmo di Karatsuba viene utilizzato per effettuare in minor tempo la moltiplicazione in binario tra due numeri. Siano $A = (a_n a_{n-1} \dots a_{k+1} a_k a_{k-1} \dots a_1 a_0)_2$ e $B = (b_n b_{n-1} \dots b_{k+1} b_k b_{k-1} \dots b_1 b_0)_2$, definiti come

$$A = (a_n a_{n-1} \dots a_{k+1})_2 * 2^k + (a_k a_{k-1} \dots a_1 a_0)_2 = A_L * 2^k + A_R \text{ e}$$

$$B = (b_n b_{n-1} \dots b_{k+1})_2 * 2^k + (b_k b_{k-1} \dots b_1 b_0)_2 = B_L * 2^k + B_R, \text{ il prodotto di } A \text{ per } B \text{ è uguale}$$

$$\begin{aligned} A * B &= (A_L * 2^k + A_R)(B_L * 2^k + B_R) = \\ &= A_L B_L * 2^{2k} + (A_L B_R + A_R B_L) * 2^k + A_R B_R. \end{aligned}$$

Capitolo 8

Rappresentazione dei numeri frazionari in binario

8.1 Codifica in virgola fissa

Viene effettuato lo stesso ragionamento dei numeri interi, ma per calcolare la parte decimale vengono usate le potenze negative (2^{-i}). Esempio:

$$6,75_{10} = 4 + 2 + 0,5 + 0,25 = 2^2 + 2^1 + 2^{-1} + 2^{-2} \rightarrow 6,75_{10} = 0110,1100_2$$

Dove la parte prima della virgola è la parte intera, mentre quella dopo è la parte decimale. Per i numeri negativi si può usare sempre il complemento a 2:

$$2,375_{10} = 2 + 0,25 + 0,125 = 2^1 + 2^{-2} + 2^{-3} \rightarrow 2,375_{10} = 0010,0110_2$$

$$0010,0110_2 \rightarrow 1101,1001 +$$

$$1$$

$$1101,1010_2 = -2,375_{10}$$

8.2 Codifica in virgola mobile

È possibile scrivere numeri molto grandi o molto piccoli attraverso la notazione esponenziale:

$$185740000000 = 1,8574 * 10^{10}$$

$$0,0000000029451 = 2,9451 * 10^{-9}$$

È possibile codificare in binario questi numeri attraverso la codifica in virgola mobile (secondo lo standard IEEE 754), in particolare attraverso la codifica a 32 bit e a 64 bit. In entrambe le notifiche, se necessario, si possono inserire degli zero subito dopo la mantissa per arrivare a 32/64 bit, inoltre, il numero ottenuto è convertibile in esadecimale, in modo da semplificare la lettura del numero.

8.2.1 Codifica a 32 bit

- Il primo bit è il bit di segno (0 per il segno + e 1 per il segno -);
- I successivi 8 bit rappresentano l'esponente e sono calcolati in eccesso (si aggiunge 127 all'esponente del numero scritto in notazione esponenziale);
- Gli altri 23 bit sono usati per la mantissa (il bit più significativo della mantissa viene dimenticato).

Esempio:

$$\begin{aligned} 185740000000_{10} &= 1,8574 * 10^{10} \\ 18574_{10} &= 1001000100011110_2 \\ 10 + 127 &= 137_{10} = 10001001_2 \\ 185740000000_{10} &= 01000100100100010001110000000000_2 = \\ &= 44911C00_H \end{aligned}$$

8.2.2 Codifica a 64 bit

- Il primo bit è il bit di segno (0 per il segno + e 1 per il segno -);
- I successivi 11 bit rappresentano l'esponente e sono calcolati in eccesso (si aggiunge 1023 all'esponente del numero scritto in notazione esponenziale);
- Gli altri 52 bit sono usati per la mantissa (il bit più significativo della mantissa viene dimenticato).

Esempio:

[illegible]

Capitolo 9

L'algebra booleana

9.1 Struttura delle formule

9.1.1 I simboli

L'algebra booleana proposizionale usa tre tipologie di simboli:

- Variabili: A, B, C, \dots ;
- Costanti:
 - Vero: 1 [true];
 - Falso: 0 [false];
- Operatori:
 - Negazione: \neg [not];
 - Congiunzione logica: \cdot [and];
 - Disgiunzione logica: $+$ [or];
 - Disgiunzione esclusiva: \oplus [xor].

9.1.2 La sintassi

La sintassi delle formule dell'algebra booleana è la seguente:

- Variabili e costanti sono formule;
- Se X è una formula, allora anche \bar{X} è una formula;
- Se X e Y sono formule, allora anche $X \cdot Y$, $X + Y$, $X \oplus Y$ sono formule;
- Nient'altro è una formula.

9.1.3 I postulati

Le formule della dell'algebra booleana si basa su 3 postulati:

- *Algebra binaria*: $B = 0$ se $B \neq 1$, $B = 1$ se $B \neq 0$;
- *Not*: $\bar{0} = 1$, $\bar{1} = 0$;
- *And/or*:
 - $0 \cdot 0 = 0$ e $0 + 0 = 0$;
 - $1 \cdot 1 = 1$ e $1 + 1 = 1$;
 - $0 \cdot 1 = 1 \cdot 0 = 0$ e $0 + 1 = 1 + 0 = 1$.

9.2 I teoremi ad una variabile

Sia A una variabile booleana:

- *Identità*: $A \cdot 1 = A$, $A + 1 = A$;
- *Valore assorbente*: $B \cdot 0 = 0$, $B + 1 = 1$;
- *Idempotenza*: $AA = A$, $A + A = A$;
- *Involuzione*: $\bar{\bar{A}} = A$;
- *Complementari*: $A\bar{A} = 0$, $A + \bar{A} = 1$.

9.3 I teoremi a più variabili

Siano A, B, C, D delle variabili booleane:

- *Commutatività*: $AB = BA$, $A + B = B + A$;
- *Associatività*: $(AB)C = A(BC)$, $(A + B) + C = A + (B + C)$;
- *Distributività*: $AB + AC = A(B + C)$, $(A + B)(A + C) = A + BC$;
- *Assorbimento*: $A(A + B) = A$, $A + AB = A$, $A(\bar{A} + B) = AB$, $A + \bar{A}B = A + B$;
- *Combinazione*: $AB + A\bar{B} = A$, $(A + B)(A + \bar{B}) = A$;
- *Consenso*: $AB + \bar{A}C + BC$, $AB + \bar{A}C$, $(A + B)(\bar{A} + C)(B + C) = (A + B)(\bar{A} + C)$.

9.4 Il teorema di De Morgan

Siano A_0, A_1, \dots, A_n delle variabili booleane:

- $\overline{A_0 A_1 \cdot \dots \cdot A_n} = \bar{A}_0 + \bar{A}_1 + \dots + \bar{A}_n$;
- $\overline{A_0 + A_1 + \dots + A_n} = \bar{A}_0 \bar{A}_1 \cdot \dots \cdot \bar{A}_n$.

9.5 Il codice di Gray

Il codice di Gray è un codice binario avente come particolarità la variazione di un solo bit da un elemento ad un altro. Infatti, presi due numeri binari $A = (a_n a_{n-1} \dots a_1 a_0)_2$ e $B = (b_n b_{n-1} \dots b_1 b_0)_2$, essi sono adiacenti nel codice di Gray se $\sum_{i=0}^n |a_i - b_i| = 1$.

9.6 La mappa di Karnaugh

La mappa di Karnaugh è una particolare tabella di verità che consente immediatamente di operare alcune semplificazioni, e risultano applicabili efficacemente solo a funzioni con al più 5-6 variabili. Si basa sul realizzare una tabella $k * h$ (dove $k + h$ è il numero di variabili usate), con le sequenze di assegnazioni dei valori alle variabili ordinate secondo il codice di Gray. Inseriti i valori, si raggruppano gli uno in rettangoli aventi area (ossia contengono un numero di 1) uguale ad una potenza di 2, e per ciascun raggruppamento identifichiamo le variabili la cui assegnazione non varia di valore, ottenendo dei prodotti. L'espressione finale si ottiene sommando i prodotti precedentemente trovati. Importante da notare è che gli 1 possono essere usati in raggruppamenti multipli, inoltre, la tabella può essere distribuita su una sfera: ciò significa che gli estremi della tabella sono connessi tra di loro, permettendo ulteriori raggruppamenti.

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$AB \backslash CD$	00	01	11	10
00	1	1	1	1
01	1	0	1	1
11	1	1	1	1
10	1	0	0	1

- Raggruppamento [0000, 0010, 0100, 0110, 1100, 1110, 1000, 1010]: l'unico valore che non cambia è D (rimane sempre uguale a 0) $\Rightarrow \bar{D}$;
- Raggruppamento [0000, 0001, 0011, 0010]: i valori che non cambiano sono A e B (rimangono sempre uguali a 0) $\Rightarrow \bar{A}\bar{B}$;
- Raggruppamento [1100, 1101, 1111, 1101]: i valori che non cambiano sono A e B (rimangono sempre uguali ad 1) $\Rightarrow AB$;
- Raggruppamento [0011, 0010, 0111, 0110]: i valori che non cambiano sono A e C (rimangono sempre uguali rispettivamente 0 ed 1) $\Rightarrow A\bar{C}$;







- Raggruppamento [0111, 0110, 1111, 1110]: i valori che non cambiano sono B e C (rimangono sempre uguali ad 1) $\Rightarrow BC$.


Pertanto $Y = \bar{D} + \bar{A}\bar{B} + AB + A\bar{C} + BC$.

Capitolo 10

Le porte logiche

Di seguito è riportata una tabella con tutte le porte logiche di base, con relativa formula e tabella di verità.

Porta	Simbolo	Formula/descrizione	Tabella di verità										
Porta NOT		Nega il segnale in ingresso ($Y = \bar{A}$).	<table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	Y	0	1	1	0				
A	Y												
0	1												
1	0												
Porta AND		Il segnale in uscita è vero se e solo se entrambi i segnali in ingresso sono veri ($Y = AB$).	<table><tr><th>AB</th><th>Y</th></tr><tr><td>0 0</td><td>0</td></tr><tr><td>0 1</td><td>0</td></tr><tr><td>1 0</td><td>0</td></tr><tr><td>1 1</td><td>1</td></tr></table>	AB	Y	0 0	0	0 1	0	1 0	0	1 1	1
AB	Y												
0 0	0												
0 1	0												
1 0	0												
1 1	1												
Porta NAND		Il segnale in uscita è vero se e solo se entrambi i segnali in ingresso non sono entrambi veri veri ($Y = \overline{AB}$).	<table><tr><th>AB</th><th>Y</th></tr><tr><td>0 0</td><td>1</td></tr><tr><td>0 1</td><td>1</td></tr><tr><td>1 0</td><td>1</td></tr><tr><td>1 1</td><td>0</td></tr></table>	AB	Y	0 0	1	0 1	1	1 0	1	1 1	0
AB	Y												
0 0	1												
0 1	1												
1 0	1												
1 1	0												
Porta OR		Il segnale in uscita è vero se e solo se almeno uno dei due i segnali in ingresso è vero ($Y = A + B$).	<table><tr><th>AB</th><th>Y</th></tr><tr><td>0 0</td><td>0</td></tr><tr><td>0 1</td><td>1</td></tr><tr><td>1 0</td><td>1</td></tr><tr><td>1 1</td><td>1</td></tr></table>	AB	Y	0 0	0	0 1	1	1 0	1	1 1	1
AB	Y												
0 0	0												
0 1	1												
1 0	1												
1 1	1												
Porta NOR		Il segnale in uscita è vero se e solo se entrambi i segnali in ingresso sono falsi ($Y = \overline{A + B}$).	<table><tr><th>AB</th><th>Y</th></tr><tr><td>0 0</td><td>1</td></tr><tr><td>0 1</td><td>0</td></tr><tr><td>1 0</td><td>0</td></tr><tr><td>1 1</td><td>0</td></tr></table>	AB	Y	0 0	1	0 1	0	1 0	0	1 1	0
AB	Y												
0 0	1												
0 1	0												
1 0	0												
1 1	0												
Porta XOR		Il segnale in uscita è vero se e solo se i due segnali in ingresso sono diversi ($Y = A \oplus B$).	<table><tr><th>AB</th><th>Y</th></tr><tr><td>0 0</td><td>0</td></tr><tr><td>0 1</td><td>1</td></tr><tr><td>1 0</td><td>1</td></tr><tr><td>1 1</td><td>0</td></tr></table>	AB	Y	0 0	0	0 1	1	1 0	1	1 1	0
AB	Y												
0 0	0												
0 1	1												
1 0	1												
1 1	0												

Porta XNOR		Il segnale in uscita è vero se e solo se i due segnali in ingresso sono uguali ($Y = \overline{A \oplus B}$).	AB	Y
			0 0	1
			0 1	0
			1 0	0
			1 1	1

10.1 Equivalenza di porte (con sole porte NAND e NOR)

È possibile realizzare le porte NOT, AND e OR utilizzando solo porte NAND o NOR, come mostrato qui sotto:

- Porta NOT:

- Porta NOT realizzata con le porte NAND:



A	$Y = \overline{AA}$
0	1
1	0

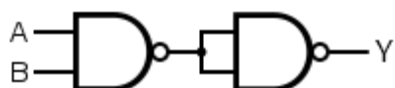
- Porta NOT realizzata con le porte NOR:



A	$Y = \overline{A + A}$
0	1
1	0

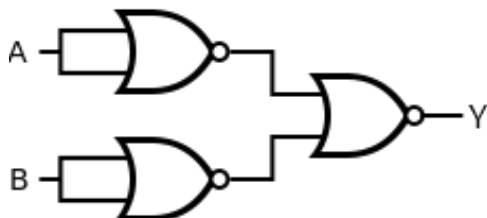
- Porta AND:

- Porta AND realizzata con le porte NAND:



A	B	$Y = \overline{\overline{AB}}$
0	0	0
0	1	0
1	0	0
1	1	1

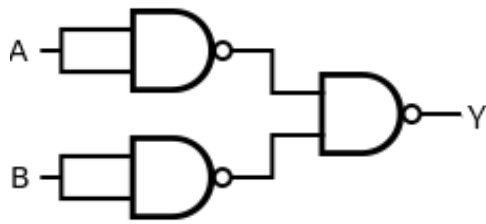
- Porta AND realizzata con le porte NOR:



A	B	$Y = \overline{\overline{A + B}}$
0	0	0
0	1	0
1	0	0
1	1	1

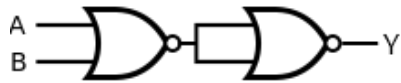
- Porta OR:

- Porta OR realizzata con le porte NAND:



A	B	$Y = \overline{\overline{A} \cdot \overline{B}}$
0	0	0
0	1	1
1	0	1
1	1	1

- Porta OR realizzata con le porte NOR:



A	B	$Y = \overline{\overline{A} + \overline{B}}$
0	0	0
0	1	1
1	0	1
1	1	1

Capitolo 11

I blocchi funzionali

11.1 Decoder

Un decoder presenta n ingressi e 2^n uscite, e attiva una delle sue uscite a seconda della combinazione di valori in ingresso.

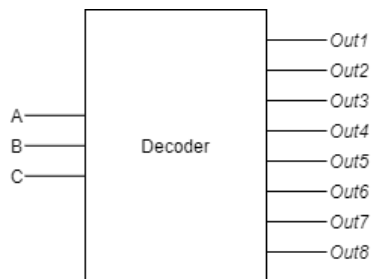


Figura 11.2: Un esempio di decoder con 3 ingressi ed 8 uscite (a destra è indicata la relativa tabella di verità).

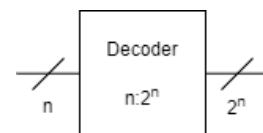


Figura 11.1: Il simbolo del decoder.

<i>A</i>	<i>B</i>	<i>C</i>	<i>Y</i>
0	0	0	<i>Out1</i>
0	0	1	<i>Out2</i>
0	1	0	<i>Out3</i>
0	1	1	<i>Out4</i>
1	0	0	<i>Out5</i>
1	0	1	<i>Out6</i>
1	1	0	<i>Out7</i>
1	1	1	<i>Out8</i>

11.2 Multiplexer

Un multiplexer presenta 2^n ingressi, 1 uscita e n selettori, ed è in grado di scegliere uno degli ingressi da mandare in uscita basandosi sul valore dei selettori.

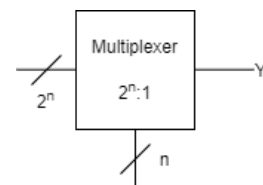
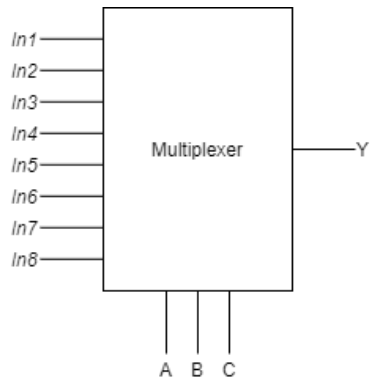


Figura 11.3: Il simbolo del multiplexer.



<i>A</i>	<i>B</i>	<i>C</i>	<i>Y</i>
0	0	0	<i>In1</i>
0	0	1	<i>In2</i>
0	1	0	<i>In3</i>
0	1	1	<i>In4</i>
1	0	0	<i>In5</i>
1	0	1	<i>In6</i>
1	1	0	<i>In7</i>
1	1	1	<i>In8</i>

Figura 11.4: Un esempio di multiplexer con 8 ingressi e 3 selettori (sotto è indicata la relativa tabella di verità).

11.3 Half adder

Un half adder presenta 2 ingressi e 2 uscite, effettuando la somma dei due ingressi e restituendo sia la somma che il riporto.

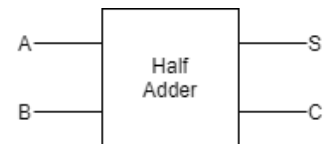
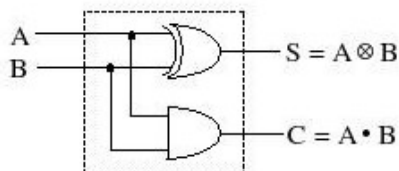


Figura 11.5: Il simbolo dell'half adder.



<i>A</i>	<i>B</i>	<i>C</i>	<i>S</i>
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figura 11.6: La struttura di un half adder (a destra è indicata la relativa tabella di verità).

11.4 Full adder

Un full adder presenta 3 ingressi e 2 uscite, effettuando la somma dei tre ingressi e restituendo sia la somma che il riporto.

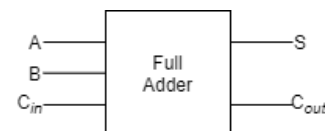


Figura 11.7: Il simbolo del full adder.

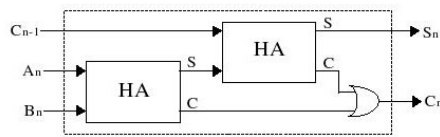


Figura 11.8: La struttura di un full adder (a destra è indicata la relativa tabella di verità).

A	B	C_{n-1}	S	C_n
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	0

11.5 Sommatore a propagazione di riporto ad onda

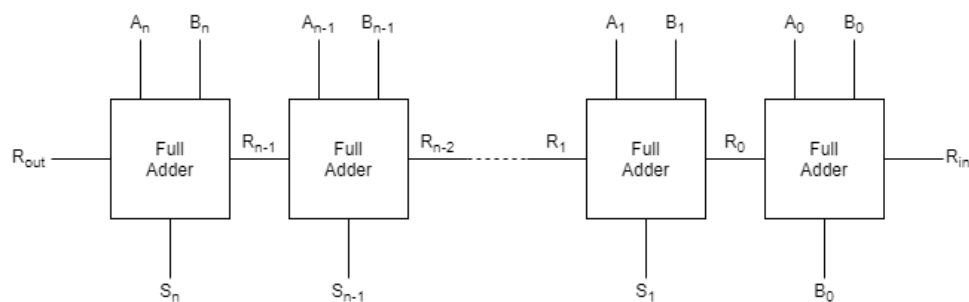


Figura 11.9: La struttura di un sommatore a propagazione di riporto ad onda.

Il metodo più semplice per realizzare un sommatore ad n bit è quello di collegare n full adder a cascata: in questo modo, l' R_{out} di uno stato costituisce l' R_{in} per il successivo. Questo tipo di sommatore è detto sommatore a propagazione di riporto ad onda (Ripple carry). Il suo problema è che impiegherà n unità di tempo per calcolare R_{out} .

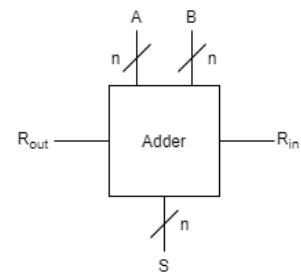


Figura 11.10: Il simbolo del sommatore a propagazione di riporto ad onda.

11.6 Sommatore/sottrattore

Il sommatore/sottrattore è un circuito combinatorio che permette la somma o la sottrazione (attraverso la conversione di uno dei due numeri in complemento a 2) di due numeri lunghi n bit. Il circuito può essere implementato utilizzando un sommatore a propagazione di riporto ad onda. Per indicare il tipo di operazione che si vuole eseguire, si usa il valore d'ingresso k , infatti:

- Se $k = 0$ allora il circuito effettuerà la somma tra i due numeri;
- Se $k = 1$ allora il circuito effettuerà la sottrazione tra i due numeri.

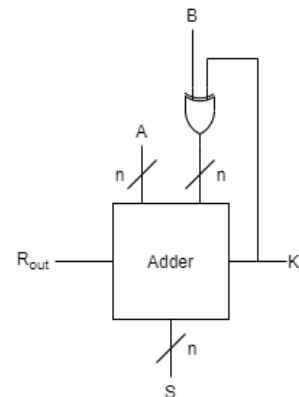


Figura 11.11: Il simbolo del sommatore/sottrattore.

11.7 Sommatore ad anticipazione di riporto

Poniamo di avere due segnali: uno di generazione (G), e uno di propagazione (P). Un sommatore i genera un riporto se questo produce un riporto in uscita a prescindere da quello in ingresso (quindi $G_i = A_i B_i$), mentre propaga un riporto se produce un riporto in uscita ogni volta ci sia un riporto in ingresso (tramite $P_i = A_i + B_i$). Da qui abbiamo che per un sommatore i il riporto in uscita è uguale a $R_{i+1} = G_i + P_i R_i$. Questo circuito è applicabile a cascata, permettendo di calcolare in anticipo il riporto in uscita ottenuto dalla somma di due numeri A e B ; tale circuito prende il nome di CLA (Carry Look-Ahead). Il CLA può essere integrato nel sommatore a propagazione di riporto ad onda: tale circuito è chiamato CLA Adder (Carry look-Ahead Adder).

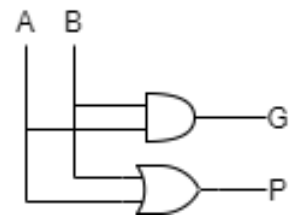


Figura 11.12: Il circuito per il calcolo dei segnali G e P .

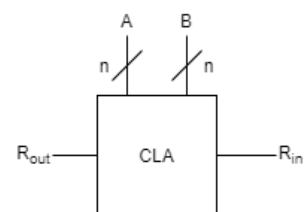


Figura 11.13: Il simbolo del CLA.

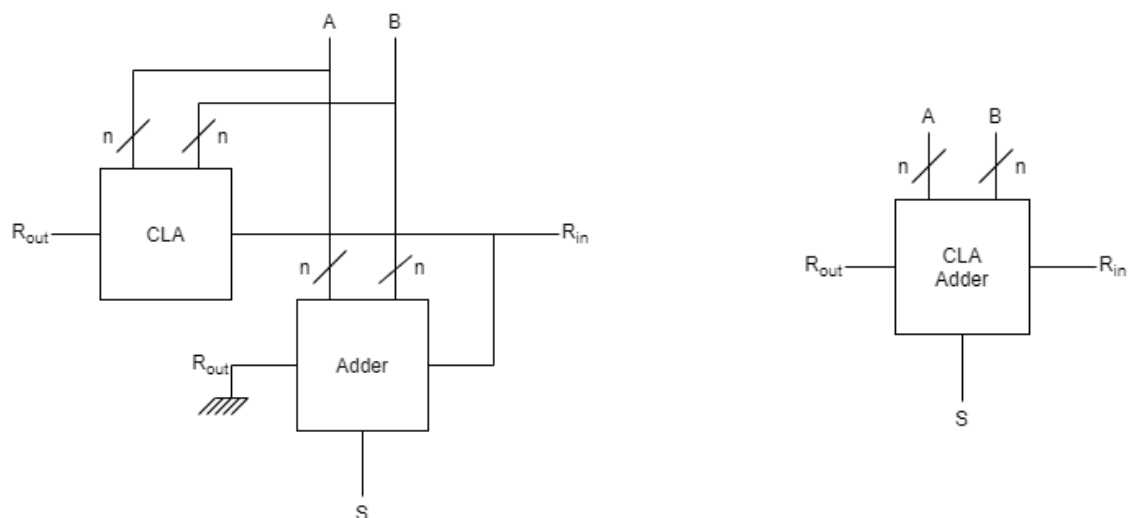


Figura 11.14: A sinistra: il circuito del CLA Adder. A destra: il simbolo del CLA Adder

11.8 Lo shifter

Lo shifter è circuito che, preso in input una stringa di n bit e un bit C , restituisce la sequenza di 1 posto a destra (se $C = 1$) o a sinistra (se $C = 0$). Da notare che shiftare una stringa di n bit a destra o a sinistra di m bit significa dividere o moltiplicare quel numero per 2^m .

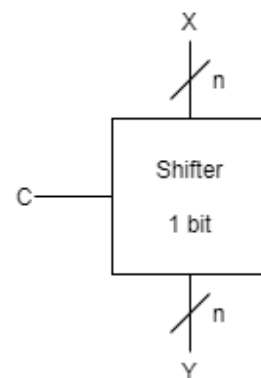


Figura 11.15: Il simbolo dello shifter

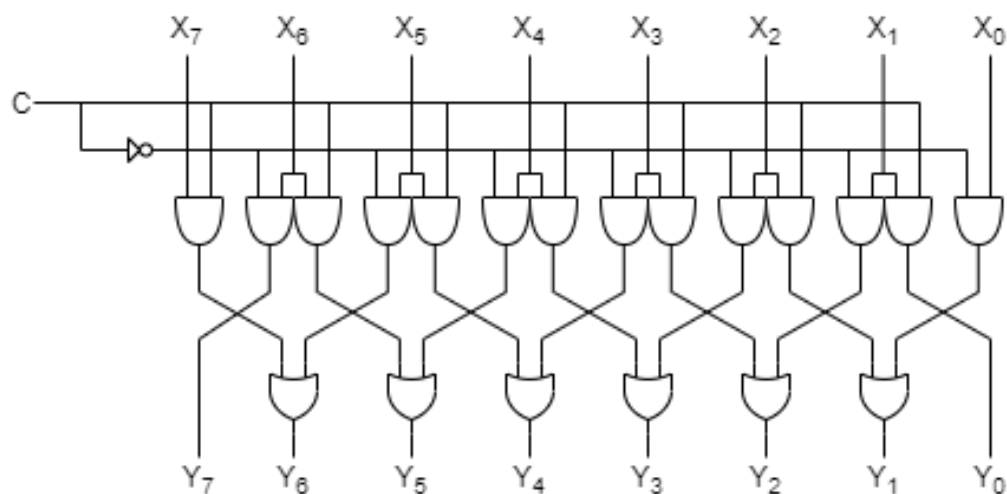


Figura 11.16: Il circuito di uno shifter ad 8bit

Capitolo 12

I circuiti sequenziali

12.1 Latch SR

Un latch SR ha due ingressi e due uscite, e permette di memorizzare un valore. Esistono due tipi di latch SR: attivo alto (creato con porte nor) e attivo bassi (creato con porte nand).

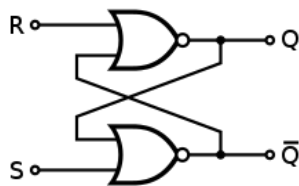


Figura 12.2: Il circuito del latch SR.

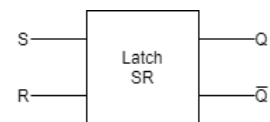


Figura 12.1: Il simbolo del latch SR.

S	R	Q	\bar{Q}
0	0	Q_P	\bar{Q}_P
0	0	0	1
0	1	1	0
0	1	0	0

12.2 Latch D

Un latch D ha due ingressi, uno per i dati e uno per il clock, e due uscite. Questa modifica del latch permette di eliminare il caso paradossale $Q = \bar{Q}$.

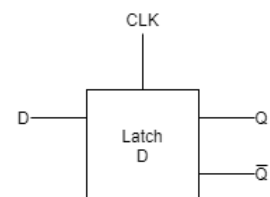


Figura 12.3: Il simbolo del latch D.

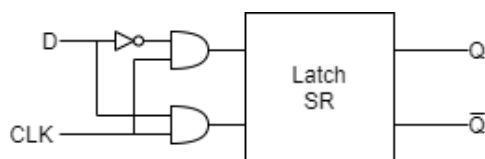


Figura 12.4: Il circuito del latch D.

S	R	Q	\bar{Q}
0	0	Q_P	\bar{Q}_P
0	0	0	1
0	1	1	0

12.3 Flip-flop D

Un flip-flop D ha due ingressi, uno per i dati e uno per il clock, e due uscite, inoltre è formato da due latch D, concatenati fra loro, definiti come master e slave. Questa struttura genera un particolare behaviour:

- Quando il clock assume valore 0, il master registra il valore in ingresso, mentre lo slave continua a mostrare il valore precedente;
- Quando il clock assume valore 1, lo slave aggiorna i propri valori Q e \bar{Q} , mentre il master mantiene i valori precedentemente letti

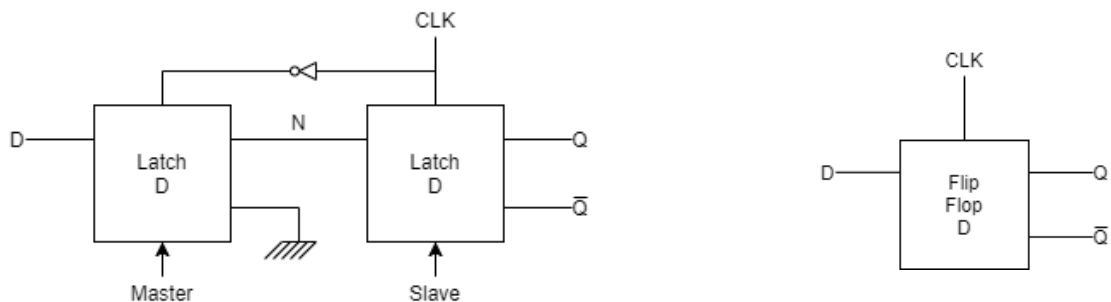


Figura 12.5: A sinistra: il circuito del flip flop D. A destra: il simbolo del flip flop D.

12.4 Registro ad n bit

Un registro ad n bit è un banco di n flip flop D che condividono lo stesso segnale di clock, in modo che tutti i bit vengano aggiornati allo stesso momento.

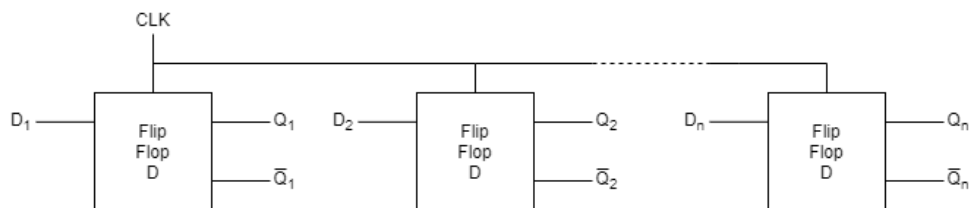


Figura 12.6: La struttura di un registro ad n bit.

12.5 Circuiti sequenziali

I circuiti sequenziali sono circuiti il cui output non dipende solo dalle variabili di input, bensì anche dalle variabili di stato interne al circuito.

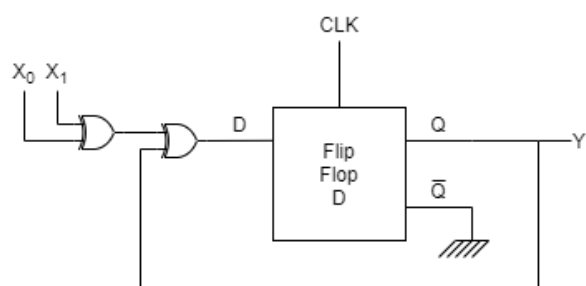


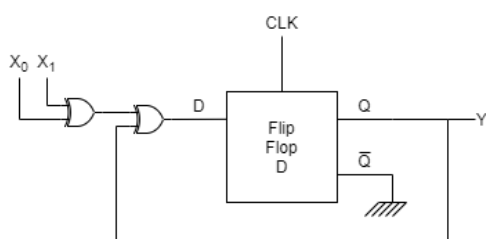
Figura 12.7: Un esempio di circuito sequenziale (a destra è presente la relativa tabella di verità, mentre sotto sono indicate le formule dei relativi valori).

Q	X_0	X_1	D	Y
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\begin{cases} Y = Q \\ D = (X_0 \oplus X_1) \oplus Q \end{cases}$$

Capitolo 13

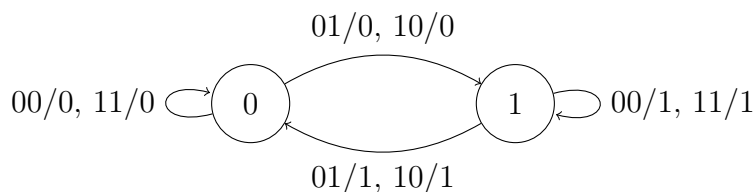
Le macchine a stati finiti



$$\begin{cases} Y = Q \\ D = (X_0 \oplus X_1) \oplus Q \end{cases}$$

Figura 13.1: Un esempio di macchina a stati finiti (a destra è rappresentato il suo diagramma di stato).

Le macchine a stati finiti, dette anche finite state machines (FSM), sono formate da 2^k stati, dove k è il numero di registri utilizzati. Dati due stati ed un collegamento tra di loro, questo si legge nel seguente modo "Letto X in input nello stato S_0 , la macchina passa nello stato S_1 restituendo Y " (esempio: letto 01 nello stato 0, la macchina passa nello stato 1 restituendo 0). S_0 ed S_1 sono rispettivamente i valori degli stati Q e D ; X sono i valori di input, mentre Y sono i valori di output.



13.1 Macchine alla Moore e macchine alla Mealy

Le macchine alla Moore sono macchine in cui l'output dipende solo dallo stato del circuito, non considerando quindi l'input. Le macchine alla Mealy sono macchine il cui output dipende sia dallo stato interno del circuito, sia dall'input. A livello grafico, nel diagramma di stato, nelle macchine alla Moore l'output è indicato nei cerchi (oltre allo stato in cui si muove la macchina), mentre nelle macchine alla Mealy l'output è indicato sugli archi (insieme, ovviamente, ai valori di input).

13.2 Fattorizzazione

Con fattorizzazione si indica la divisione di una macchina a stati complessa in macchine a stati più semplici che interagiscono tra loro, in modo che le uscite di alcune siano gli ingressi di altre.