Capitolo primo

Sistemi di numerazione

Generalità

Un calcolatore elettronico è costituito da circuiti integrati che elaborano informazioni rappresentate nel sistema di numerazione binario; questo si basa su due sole cifre: 0 e 1.

Ciò si deve alla modalità di memorizzazione dell'informazione nei dispostivi elettronici (transistor e MOSFET) interni ai circuiti integrati che possono facilmente funzionare in saturazione (valore basso di tensione: 0 logico) o in interdizione (valore alto di tensione: 1 logico).

Tali segnali si dicono **digitali** o binari perchè possono assumere due soli valori. In contrapposizione vi sono i segnali **analogici** che possono assumere tutti i valori compresi in un determinato intervallo.

1. Sistema di numerazione decimale

Un sistema di numerazione si basa su un numero finito di simboli. Tale numero prende il nome di **base**. Per questo motivo il sistema decimale, che impiega 10 simboli diversi (da 0 a 9) è noto come sistema di numerazione in base 10. Esso è stato inventato dagli arabi e si è diffuso notevolmente perchè il numero delle cifre coincide con il numero delle dita delle due mani. Esso è un sistema **pesato** e **posizionale**: è pesato perchè ogni cifra ha un proprio valore, è posizionale perchè il valore della cifra dipende anche dalla posizione occupata nel numero.

Ad esempio la cifra 1 nel numero 172 ha valore diverso rispetto a quella che compare nel numero 631.

La cifra più a sinistra si chiama **cifra più significativa** e quella più a destra si chiama **cifra meno sisgnificativa**. Si osservi che non tutti i sistemi di numerazione sono posizionali: si pensi al sistema di numerazione romano nel quale il numero 10 si rappresenta con X, il numero 100 con C, 1000 con M, ecc. Poichè le posizioni assunte dalle cifre sono fisse.

Un numero decimale si può scomporre in forma polinomiale come somma di prodotti tra i valori intrinseci delle cifre per una potenza di 10 con esponenti decrescenti da n-1 fino a 0 con nel caso di un numero costituito da n cifre.

Ad esempio il numero 271 è costituito da 2 centinaia, 7 decine ed 1 unità per cui si può scomporre in:

$$271 = 2\cdot10^2 + 7\cdot10^1 + 1\cdot10^0$$

È possibile scomporre anche numeri non interi continuando ad esprimere gli esponenti del 10 con numeri decrescenti negativi come nel seguente esempio:

$$271.56 = 2\cdot10^2 + 7\cdot10^1 + 1\cdot10^0 + 5\cdot10^{-1} + 6\cdot10^{-2}$$

Nel sistema di numerazione decimale le quattro operazioni si possono eseguire applicando delle semplici regole a differenza del sistema romano che richiedeva l'impiego di persone esperte.

2. Sistema di numerazione binario

Il sistema di numerazione binario è di tipo pesato e posizionale. Si basa su due sole cifre indicate con 0 e 1. Ciascuna delle due cifre prende il nome di bit.

Anch'essi si possono scomporre in somma di prodotti tra i bit e le poptenze del 2 con esponente decrescente da n-1 a 0 con un numero ad n bit.

Ad esempio il numero binario a 4 bit 1101 si scompone così:

$$1101_2 = 1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 = 8 + 4 + 0 + 1 = 13_{10}$$

La scomposizone polinomiale rappresenta un metodo per la conversione di un numero dal sistema binario in decimale.

In tabella 1 si mostrano le prime 20 potenze positive e negative del 2:

Tab. 1

n	2^n	2^{-n}
0	1	1
1	2	0,5
2	4	0,25
3	8	0,125
4	16	0,0625
5	32	0,03125
6	64	0,015625
7	128	0,0078125
8	256	0,00390625
9	512	0,001953125
10	1024	0,0009765625
11	2048	0,00048828125
12	4096	0,000244140625
13	8192	0,0001220703125
14	16384	0,00006103515625
15	32768	0,000030517578125
16	65536	0,0000152587890625
17	131072	0,00000762939453125
18	262144	0,000003814697265625
19	524288	0,0000019073486328125
20	1048576	0,00000095367431640625

Nella rappresentazione di numeri binari gli zeri non significativi posti a sinistra del numero possono essere soppressi. Ad esmpio il numero 6 si esprime in binario con 110.

I numeri potenza del due (1, 2, 4, 8, 16, ecc.) si rappresentano in binario con uno seguito da tanti zeri quanto è l'esponnte da dare al due per otteneretale potenza. Ad esempio: $8 = 2^3 = 1000_2$.

In tabella 2 si mostrano le configurazioni dei primi 32 numeri binari.

Tab. 2

n	B_4	B_3	B_2	B_1	B_0	n	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	16	1	0	0	0	0
1	0	0	0	0	1	17	1	0	0	0	1
2	0	0	0	1	0	18	1	0	0	1	0
3	0	0	0	1	1	19	1	0	0	1	1
4	0	0	1	0	0	20	1	0	1	0	0
5	0	0	1	0	1	21	1	0	1	0	1
6	0	0	1	1	0	22	1	0	1	1	0
7	0	0	1	1	1	23	1	1	0	1	1
8	0	1	0	0	0	24	1	1	0	0	0
9	0	1	0	0	1	25	1	1	0	0	1
10	0	1	0	1	0	26	1	1	0	1	0
11	0	1	0	1	1	27	1	1	0	1	1
12	0	1	1	0	0	28	1	1	1	0	0
13	0	1	1	0	1	29	1	1	1	0	1
14	0	1	1	1	0	30	1	1	1	1	0
15	0	1	1	1	1	31	1	1	1	1	1

Anche i numeri binari possono essere non interi. Ad esempio il numero 101.101 si può scomporre così:

$$101.101_2 = 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 + 1\cdot2^{-1} + 0\cdot2^{-2} + 1\cdot2^{-3} = 4 + 1 + 0.5 + 0.125 = 5.625_{10}$$

Quanto finora asserito per i numeri decimali e binari vale anche per numeri espressi in una base diversa, genericamente indicata con b.

Sia $C_{n-1}C_{n-2}\ldots C_1C_0$ un numero espresso nella base b costituito dalle n cifre:

$$C_{n-1},\ C_{n-2},\ \ldots\ C_1 \text{ e } C_0$$

Tale numero è scomponibile in:

$$(C_{n-1}C_{n-2}\ldots C_1C_0)_n = C_{n-1}\cdot b^{n-1} + C_{n-2}\cdot b^{n-2}\ldots.+C_1\cdot b^1 + C_0\cdot b^0$$

La somma dei termini a secondo membro della precedente uguaglianza rappresenta il corrispondente numero decimale. Ad esempio il numero 177 in base 8 (sistema ottale costituito da 8 simboli che vanno da 0 a 7) è scomponibile in:

$$177_8 = 1\cdot8^2 + 7\cdot8^1 + 7\cdot8^0 = 64 + 56 + 7 = 127_{10}$$

2.1. Conversione dal sistema decimale al binario

Per convertire un numero dal sistema decimale al sistema binario si deve applicare la regola dei resti che si ottengono dalla divisione intera per 2 ripetuta fino ad ottenere un resto nullo. Ad esempio, per convertire in binario il numero decimale 13 si devono svolgere i seguenti passi:

1) si divide 13 per 2 e si considera la parte intera del quoto (6) che si scrive sotto il 13 e si pone il resto della divisione (1) a destra di 13;
2) si divide 6 per 2 e si considera la parte intera del quoto (3) che si scrive sotto il 6 e si pone il resto della divisione (0) a destra del 6;
3) si divide 3 per 2 e si considera la parte intera del quoto (1) che si scrive sotto il 3 e si pone il resto della divisione (1) a destra del 3;
4) si divide 1 per 2 e si scrive il quoto (0) sotto l'1 ed il resto (1) a destra dell'1.

L'operazione delle divisioni successive ha termine perché si è raggiunto il quoto uguale a zero. Il numero binario richiesto si ottiene dall'insieme dei resti presi in senso inverso, cioè dal basso verso l'alto:

passi					
	1	13	1	LSB	
	2	6	0		
	3	3	1		
	4	1	1	MSB	$13_{10} = 1101_2$
		0			

Per convertire in binario un numero decimale non intero si procede con il metodo delle moltiplicazione ripetute per due. Per convertire in binario il numero 0.375 si procede come segue:

1) si moltiplica 0.375 per due e si inserisce la parte intera del prodotto (0) a destra di 0.375 e la parte decimale del prodotto (0.75) sotto 0.375;
2) si moltiplica 0.75 per due e si inserisce la parte intera del prodotto (1) a destra di 0.75 e la parte decimale (0.5) sotto 0.75;
3) si moltiplica 0.5 per due e si inserisce la parte intera del prodotto (1) a destra di 0.5 e la parte decimale (0) sotto 0.5.

Il procedimento ha termine perché la parte decimale dell'ultimo prodotto ottenuto è nullo.

Si fa notare che molto spesso, a differenza dell'esempio precedente, il numero di bit necessari per convertire un numero decimale intero compreso tra 0 e 1 può essere illimitato. In tal caso il numero di moltiplicazioni per due deve essere arrestato dopo un numero finito di passi che può essere aumentato per migliorare la precisione e con essa aumentata il numero di bit da utilizzare (dopo il punto).

1°	0.375	0	MSB	
2°	0.75	1		
3°	0.5	1	LSB	
	0			$0.375_{10} = 0.011_2$

Esempio

Convertire 0.2_{10} in base 2.

0.2	0
0.4	0
0.8	1
0.6	1
0.2	0
0.4	0
0.8	1
0.6	1
0.2	..
...	..

sequenza periodica

$0.2_{10} = 0.001100110011\ldots_2$

Se si approssima la conversione con i primi quattro bit dopo il punto, si ottiene:

$$0.0011_2 = 0\cdot2^{-1} + 0\cdot2^{-2} + 1\cdot2^{-3} + 1\cdot2^{-4} = 0.125 + 0.0625 = 0.1875$$

L'errore relativo in percentuale commesso vale:

$$e_r\% = (0.2 - 0.1875)\cdot100/0.2 = 6.25\%$$

In generale, detto *N* il numero decimale intero da convertire in binario, il numero di bit necessari per la sua rappresentazione è pari all'esponente *n* del due che soddisfa la disequazione:

$$2^{n-1} < N < 2^n$$

2.2 Operazioni con i numeri binari

Le regole per effettuare le operazioni aritmetiche sono le stesse che si applicano con i numeri decimali poiché sia il sistema di numerazione binario che decimale sono di tipo posizionale.

ADDIZIONE

La somma tra due numeri binari si effettua sommando tra loro i bit di stesso peso più l'eventuale riporto proveniente dalla somma dei bit di peso immediatamente più basso ripetendo il procedimento dai bit **LSB** verso i bit **MSB**. Per meglio comprendere il procedimento si esegue la somma di due bit.

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 10$ (0 con riporto di 1)

Si noti che l'ultima somma, nel sistema decimale, fornisce come risultato 2 che, in binario, si esprime con 10. Per effettuare la somma tra due numeri binari a più bit, tuttavia, occorre considerare, come già anticipato, il riporto proveniente dalla somma dei bit di peso immediatamente più basso.

Ad esempio:

	DECIMALE	BINARIO
A	44 +	101100 +
B	27 =	11011 =
	71	1000111

Nella tabella 3 si riportano tutte le combinazioni che si possono verificare quando si sommano due bit più il riporto precedente. Con A_i e B_i si indicano i bit di peso 2^i dei numeri binari *A* e *B* rispettivamente, con C_{i-1} si indica il riporto proveniente dalla somma dei bit di peso 2^{i-1} con S_i e C_i si indicano la somma e il riporto generati:

Tab. 3

A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

SOTTRAZIONE

Per effettuare la differenza tra due numeri binari si applicano le stesse regole della sottrazione tra due numeri decimali. Conviene,come già visto per l' addizione, riportare i risultati della differenza tra i numeri binari ad un bit.

0 – 0 = 0

0 – 1 = 1 con prestito da sinistra

1 – 0 = 1

1 – 1 = 0

In tabella 4 si riportano tutte le combinazioni che si possono verificare quando si sottraggono due bit corrispondenti di due numeri binari tenendo presente l' eventuale prestito effettuato. Con P_{i-1} si indica il

prestito subito dal minuendo Ai durante la sottrazione dei bit di peso 2^{i-1}, con Di si indica l' esito della differenza e con Pi si indica il prestito richiesto al minuendo A_{i+1}.

Tab. 4

A_i	B_i	P_{i-1}	D_i	P_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Esempio

DECIMALE	BINARIO
A 44 -	101100 -
B 27 =	11011 =
17	10001

La sottrazione può essere eseguita anche applicando il metodo del complemento a 2. In tal caso la sottrazione è ricondotta ad una somma. È necessario, però, anticipare il concetto di complemento alla base.

COMPLEMENTO ALLA BASE

Qualunque sia la base b di un numero N espresso con n cifre, si dice complemento alla base il numero che si ottiene sottraendo N dalla potenza b^n, cioè: b^n–N = complemento a b del numero N.

Nel sistema di numerazione decimale si parla di complemento a 10, in quello binario di complemento a 2, ecc.

Esempio

Il complemento a 10 di 53 è: $10^2 – 53 = 100 – 53 = 47$

Si può ottenere lo stesso risultato effettuando il complemento a 9 di ciascuna cifra e sommando 1 al risultato ottenuto.

Il complemento a 9 di una cifra è la differenza tra 9 e la cifra considerata. Nel caso precedente si ha:

complemento a 9 di 5: 9 – 5 = 4
complemento a 9 di 3: 9 – 3 = 6
complemento a 9 di 53: 46
complemento a 10 di 53 = (complemento a 9 di 53 + 1) : 46 + 1 = 47

Con la tecnica del complemento alla base è possibile ottenere la differenza tra due numeri *A* e *B* eseguendo la somma tra *A* e il complemento alla base di *B* sopprimendo la cifra più significativa ottenuta. Infatti:

$$47 - 20 = 27$$

Con la tecnica del complemento a 10 si ha:

$$47 + (100 - 20) = 47 + 80 = \underline{1}27 \text{ da cui: } 47 - 20 = 27$$

La cifra 1 sottolineata si sopprime. Il numero risultante (27) è la differenza. Analogamente, il complemento a 2 di un numero binario *N* a *n* cifre sarà:

$$N_2 = 2^n - N$$

Ad esempio, se *N* = 101 il complemento a 2 vale:

$$N_2 = 2^3 - N = 1000 - 101 = 011$$

Si definisce complemento ad 1 e si indica con N_1 il numero binario che si ottiene sottraendo una unità al complemento a 2:

$$N_1 = N_2 - 1$$

Riprendendo l'esempio precedente, se *N* = 101 con complemento a 2 N_2 = 011 si ha:

$$N_1 = N_2 - 1 = 011 - 1 = 010$$

Il complemento a 1 di un numero binario si può ricavare semplicemente complementando tutti i bit del numero dato, cioè 0 si cambia in 1 e, viceversa, 1 si cambia in 0. Ad esempio:

$N = 1101$ si ha: $N_1 = 0010$

Per ottenere il complemento a 2 basta sommare 1 al complemento ad uno del *N* assegnato.

Ad esempio, se:

$N = 11010$ si ha : $N_1 = 00101$; $N_2 = N_1 + 1 = 00101 + 1 = 00110$

Il complemento a 2 di un binario si può determinare anche mediante la seguente regola:

osservando il numero binario *N* da destra verso sinistra, si lasciano inalterati tutti i bit uguali a 0 incontrati e il primo bit uguale a 1; si complementano tutti i successivi bit a sinistra. Ad esempio:

$N = 10100100$ si ha: $N_2 = 01011100$

Sottrazione mediante i complementi

L'uso del completamento ad 1 e a 2 consente la realizzazione dell'operazione di sottrazione mediante una operazione di addizione; ciò permette, come si vedrà nei capitoli successivi, di utilizzare un unico circuito integrato (sommatore) per eseguire sia addizioni che sottrazioni.

Supponiamo di voler realizzare la differenza tra due numeri *A* e *B* entrambi positivi; si può usare la tecnica che di seguito si descrive.

È facile verificare che la differenza tra *A* e *B* si ottiene sommando ad *A* il complemento a due di *B.* Se il bit di riporto complessivo è 1 il risultato è corretto. Se invece vale 0 si deve effettuare un ulteriore complemento a 2 per determinare l'effettivo risultato in valore assoluto.

Esempio

DECIMALE	BINARIO	BINARIO con complemento a 2 del sottraendo
12-	1100 -	1100 +
9 =	1001 =	0111 =
3	0011	10011

Il bit di riporto sottolineato vale 1 e ciò indica che il risultato è corretto e tale bit va trascurato.

MOLTIPLICAZIONE

Anche per la moltiplicazione tra numeri binari valgono le regole note che si applicano per la moltiplicazione fra numeri decimali. I prodotti aritmetici tra due bit valgono:

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

Nell'esecuzione manuale della moltiplicazione si devono effettuare i prodotti tra uno dei bit del moltiplicatore ed il moltiplicando ottenendo come risultati parziali 0 o lo stesso moltiplicando a seconda se il bit moltiplicatore vale 0 o 1, rispettivamente.

Esempio

moltiplicando	11100·
moltiplicatore	101=
prodotto parziale per il bit 2^0	11100
prodotto parziale per il bit 2^2	11100--
	10001100

In pratica si riporta il moltiplicando spostato a sinistra di tante posizioni quanto è l'esponente del due o del peso del bit del moltiplicatore che vale 1. Si tralasciano gli zeri del moltiplicatore. L'operazione è, quindi ricondotta ad una serie di somme del moltiplicando incolonnato in modo opportuno. Il prodotto tra un numero binario a *n* bit con uno a *m* bit è costituito al più da *n* + *m* bit.

DIVISIONE

La divisione binaria è analoga a quella decimale ma risulta più semplice poiché l'esatto quoziente parziale ha due soli possibili valori (0,1) e quindi non si può sbagliare.

A titolo d' esempio consideriamo la seguente divisione: 30 : 5 = 6

```
11110 : 101
101    110
=101
 101
   0
```

La divisione è, quindi, riconducibile ad una serie di sottrazioni successive tre il dividendo e il divisore opportunamente incolonnato.

3. Sistema di numerazione esadecimale

È un sistema di numerazione pesato e posizionale a base 16 che utilizza 16 simboli che vanno da 0 a 9 e da A a F. Cioè:

valore decimale: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

valori esadecimali: 0 1 2 3 4 5 6 7 8 9 A B C D E F

Si noti che le prime 10 cifre esadecimali coincidono con le cifre del sistema di numerazione decimale mentre le ultime sei sono le prime sei lettere maiuscole dell' alfabeto.

Esempi di numeri esadecimali sono:

7A, 10, FF, 1B7, C0CA

In decimale essi valgono:

$$(7A)_{16} = 7\cdot16^1 + 10\cdot16^0 = 112 + 10 = 122$$

$$(10)_{16} = 1\cdot16^1 + 0\cdot16^0 = 16$$

$$(FF)_{16} = 15\cdot16^1 + 15\cdot16^0 = 240 + 15 = 255$$

$$(1B7)_{16} = 1\cdot16^2 + 11\cdot16^1 + 7\cdot16^0 = 256 + 176 + 7 = 439$$

$$(C0CA)_{16} = 12\cdot16^3 + 0\cdot16^2 + 12\cdot16^1 + 10\cdot16^0 = 49152 + 192 + 10 = 49354$$

3.1 Conversione da esadecimale a binario

Poiché $16 = 2^4$, un numero dato nel sistema esadecimale si converte facilmente in binario. Una cifra esadecimale si trasforma esattamente in 4 bit come si mostra in tab.5.

Tab.5

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

3.2. Conversione da binario ad esadecimale

Le informazioni memorizzate in un elaboratore elettronico sono espresse nel sistema di numerazione binario. Si è visto che per rappresentare un numero decimale occorrono numerosi bit e quindi il numero binario risulta poco leggibile. Con la conversione dal sistema binario all'esadecimale è possibile comprimere in poche cifre esadecimali un lunghissimo numero binario.

Per effettuare la conversione si divide il numero binario in gruppi di 4 bit a partire da destra. L'ultimo gruppo ottenuto con questo metodo può essere costituito da meno di 4 bit.

Si riportano alcuni esempi:

a) 1100101_2. Il numero assegnato viene suddiviso in due gruppi da 4 bit: 0110 0101. Dalla tabella 5 si ricava: $0110_2 = 6_{16}$ e: $0101_2 = 5_{16}$ per cui: $1100101_2 = 65_{16}$
b) $10111011010_2 = 5BA_{16}$
c) $101010_2 = 2A_{16}$

3. 3. Conversione da decimale ad esadecimale

La conversione da decimale ad esadecimale si può realizzare passando prima al sistema binario e poi all'esadecimale:

DECIMALE → BINARIO → ESADICIMALE

Il metodo diretto di conversione, invece, consiste nell'effettuale le divisioni intere ripetute per 16 e nel considerare i resti provenienti da tali divisioni finché il quoto finale diventa uguale a zero.

L'insieme dei resti, considerati in senso inverso, rappresenta il numero esadecimale richiesto.

Il metodo è del tutto simile a quello visto per la conversione da decimale a binario. Si illustra il metodo con un esempio.

Esempio

Si vuole convertire in esadecimale il numero decimale 282. Il risultato è: $11A_{16}$.

$282 : 16 = 17$ con resto $10\ (A)_{16}$
$17 : 16 = 1$ con resto 1
$1 : 16 = 0$ con resto 1

Si riportano due esempi svolti di conversione da decimale ad esadecimale.

4812	12
300	12
18	2
1	1
0	

$4812_{10} = 12CC_{16}$

7421	13
463	15
28	12
1	1

$7421_{10} = 1CFD_{16}$

4. Rappresentazione dei numeri nel calcolatore

Il calcolatore elabora dati sia numerici che alfanumerici e conserva i risultati nella sua memoria centrale, pronti per essere ulteriormente utilizzati. La memoria centrale è costituita da circuiti integrati a semiconduttori ed è organizzata in byte. Un byte è un insieme di 8 bit, che può assumere $2^8 = 256$ configurazioni possibili.

4.1. Numeri interi

La rappresentazione dei numeri interi nella memoria del calcolatore dipende dal tipo di applicazione che si sta eseguendo e, nel caso di un programma scritto in un linguaggio di programmazione ad alto livello, dal tipo di linguaggio utilizzato.

Alcuni linguaggi di programmazione, come il Pascal e il C, consentono la definizione di variabili o costanti numeriche intere con segno e senza segno.

Interi senza segno

I numeri interi senza segno sono considerati positivi e sono rappresentati all'interno di un "contenitore"che occupa uno o più byte. Un intero senza segna che occupa un byte, pertanto, può assumere tutti i valori compresi tra 0 e 255 poichè il massimo numero rappresentabile con 8 bit vale $2^8-1 = 255$.

Nel linguaggio C tale tipo di dato è definito col nome *unsegned char* (carattere senza segno) e nel linguaggio Pascal semplicemente *byte*.

Un intero senza segno a 16 bit può assumere tutti i valori compresi tra 0 e 2^{16}-1 = 65535 mentre un intero a 32 bit può assumere tutti i valori compresi tra 0 e 2^{32}-1 = 4.294.967.295.

Interi con segno

Molto spesso una variabile o una costante numerica intera assume un valore negativo e, in tal caso, si deve prevedere la rappresentatività sia di valori positivi che negativi.

In pratica con un byte si possono rappresentare 128 numeri negativi da -128 a -1 e 128 numeri positivi da 0 a 127. In tal caso lo zero è considerato positivo.

Con due byte si rappresentano i numeri negativi da -32768 a -1 ed i numeri positivi da 0 a 32767; con 4 byte da -2.147.483.648 a -1 e da 0 a 2.147.483.647.

I numeri positivi sono rappresentati in codice binario naturale ed i numeri negativi con la tecnica del complemento a 2. Ad esempio il numero -50 si rappresenta in un byte col codice 11001110.

Infatti: 50_{10} = 00110010_2. Il complemento a 1 vale: 11001101. Per ottenere il complemento a 2 si aggiunge 1 al complemento a 1: 11001101 + 1 = 11001110.

In tab.6 si mostrano le denominazioni degli interi con e senza segno nei vari linguaggi di programmazione, il numero di byte occupati ed il relativo intervallo di valori consentiti.

Tab.6

byte occupati	valori negativi	valori positivi	Basic	Pascal	C
1		0…255		byte	unsegned char
2		0…65535		word	unsegned
4		0…4.294.967.295			unsegned long
1	-128…-1	0…127		shortint	char
2	-32768…-1	0…32767	interi	integer	int
4	-2.147.483.648…-1	0…2.147.483.647	interi long	longint	long

5. Rappresentazione di dati alfanumerici nel calcolatore

I dati alfanumerici sono costituiti da caratteri alfanumerici maiuscoli e minuscoli, cifre numeriche da 0 a 9, segni di punteggiatura e caratteri speciali. Se questi non sono in numero superiore a 128 si può usare un codice a 7 bit con il quale associare, ad ogni combinazione binaria, un dato alfanumerico specifico.

5.1 Codice ASCII

Il codice senza dubbio più diffuso nei personal computer è il codice ASCII (American Standard Code for Information Interchange) che si basa su 7 bit. La codifica di ciascun carattere ASCII in realtà non avviene in 7 bit ma in 8. Ciascun carattere, quindi, è memorizzato in un byte. L'ottavo bit è ridondante e viene utilizzato

come bit di parità. Il bit di parità viene posizionato a 0 oppure a 1 in modo che il numero di bit che costituisce l'intero byte è pari (generazione del bit di parità pari).

L'utilità del bit di parità si manifesta quando si effettua la trasmissione del byte da un apparato trasmettitore verso un apparato ricevitore. Il circuito ricevitore possiede un circuito di controllo per verificare il bit di parità. Se durante la trasmissione un bit cambia di valore il controllore si accorge che il bit di parità da lui generato è diverso da quello ricevuto dal trasmettitore e quindi rileva la presenza di un errore anche se non gli è noto su quale bit si è verificato. Il controllo è efficace se accade un errore su un solo bit del byte. I codici da 0 a 32 ed il codice 127 sono caratteri speciali di comando alle periferiche come le stampanti.

Tab.7

DEC	HEX	CODICE	SIGNIFICATO
0	0	NUL	Nullo
1	1	SOH	Start Of Heading = Inizio intestazione
2	2	STX	Start Of Text = Inizio testo
3	3	ETX	End Of Text = Fine testo
4	4	EOT	End Of Trasmission = Fine trasmissione
5	5	ENQ	Enquiry = Richiesta
6	6	ACK	Acknowledge = Conferma
7	7	BEL	Bell = Segnale acustico
8	8	BS	Backspace = Ritorna indietro di un carattere
9	9	HT	Horizontal Tab = Tabulazione orizzontale
10	A	LF	Line Feed = Avanzamento linea
11	B	VT	Vertical Tab = Tabulazione Verticale
12	C	FF	Form Feed = Avanzamento della carta
13	D	CR	Carriage Return = Ritorno carrello
14	E	SO	Shift Out = Maiuscolo disinserito
15	F	SI	Shift In = Maiuscolo inserito
16	10	DLE	Data Link Escape = Uscita trasmissione
17	11	DC1	Direct Control 1 = Controllo periferica 1
18	12	DC2	Direct Control 2 = Controllo periferica 2
19	13	DC3	Direct Control 3 = Controllo periferica 3
20	14	DC4	Direct Control 4 = Controllo periferica 4
21	15	NAK	Negative Acknowledge = Conferma negativa
22	16	SYN	Syncronous Idle = Attesa sincronizzata
23	17	ETB	End Trasmission Block = Fine trasmissione
24	18	CAN	Cancel = Annulla
25	19	EM	End of Medium = Fine supporto
26	1A	SUB	Substitute = Sostituzione
27	1B	ESC	Escape = Uscita dal codice
28	1C	FS	Form Separator = Separazione file
29	1D	GS	Group Separator = Separatore di gruppo
30	1E	RS	Record Separator = Separazione record
31	1F	US	Unit Separator = Separazione unità
32	20	SP	Space = Spazio
127	7F	DEL	Delete = Cancellazione del carattere

Spesso i computer utilizzano il codice ASCII esteso a 8 bit per cui sono disponibili ben 256 configurazioni. Le prime 128 hanno lo stesso significato del codice ASCII originale mentre le altre sono utilizzate per la definizione dei caratteri semigrafici o di controllo per la stampante.

5.2. Cenni ad altri codici

Vi sono altri codici binari, per lo più per la rappresentazione di numeri, formulati per soddisfare particolari esigenze. Si descrive brevemente alcuni di essi per la rappresentazione di cifre decimali.

Gray = Passando da un numero al successivo cambia un solo bit.
BCD 8421 (Binary Coded Decimal) = Corrisponde al codice binario naturale.
BCD2421 (Codice Aiken) = Il bit più significativo vale 2 anziché 8.
Eccesso-3 = Il numero n ha la configurazione binaria naturale di n+3.
Codice 2 su 5 = Utilizza 5 bit di cui 2 sono uguali a 1 ed i restanti 3 sono uguali a 0.

I codici Aiken ed eccesso-3 sono autocomplementanti cioè le configurazioni equidistanti dagli estremi sono l'una il complemento a 9 dell'altra e si passa da l'una all'altra semplicemente scambiando gli zeri con gli uno.

Tab.9

n	8421	2421	E-3	GRAY	2 su 5
0	0000	0000	0011	0000	00011
1	0001	0001	0100	0001	00101
2	0010	0010	0101	0011	00110
3	0011	0011	0110	0010	01001
4	0100	0100	0111	0110	01010
5	0101	1011	1000	0111	01100
6	0110	1100	1001	0101	10001
7	0111	1101	1010	0100	10010
8	1000	1110	1011	1100	10100
9	1001	1111	1100	1101	11000