Appunti di Elettronica Digitale

Capitolo 1 - Numerazione binaria

NUMERAZIONE BINARIA

Quando aumentò l'interesse per la costruzione di apparecchiature meccaniche in grado di realizzare automaticamente le operazioni aritmetiche, specialmente le moltiplicazioni, *il sistema binario si diffuse molto rapidamente e sin dalla realizzazione dei primi calcolatori elettronici si dimostrò il più adatto ad essere trattato automaticamente.*

I simboli grafici fondamentali del sistema binario sono "**0**" e "**1**".

La conta nel sistema binario avviene secondo un criterio analogo a quello del sistema decimale: dopo aver considerato i due simboli fondamentali, si passa a considerarli nuovamente effettuando il riporto (cioè sommando 1) sulla cifra immediatamente a sinistra. Quindi, il simbolo decimale 2 sarà espresso in binario da 10 (che va letto uno-zero e non dieci), il simbolo 3 da 11, il simbolo 4 da 100 e così via.

Le cifre di un numero binario vengono chiamate "**bit**" (che sta per *BInary digiT*) ed i valori posizionali sono dati ovviamente dalle potenze di 2.

Un evidente vantaggio del sistema binario è il fatto che il numero di simboli grafici fondamentali da esso usati è piccolissimo (ci sono 2 soli simboli), il che permette di stabilire facilmente una corrispondenza biunivoca tra tali simboli e i due possibili stati di funzionamento di particolari circuiti elettronici o di alcuni supporti fisici. Viceversa, *un sicuro inconveniente di tale sistema è il fatto che servano molte cifre binarie (o molti bit) per rappresentare numeri comunque piccoli* che invece nel sistema decimale o in altri sistemi necessitano anche di un solo simbolo.

A questo proposito è opportuno fornire una relazione che permette di calcolare quanti bit serviranno per esprimere un qualsiasi numero decimale. Si verifica infatti che un qualsiasi numero intero decimale N si può rappresentare da un numero p di bit tale che valga la seguente relazione:

$$\boxed{2^{p-1} \le N < 2^{p}}$$

Sfruttando questa relazione, si trova facilmente che i numeri decimali fino a 2 cifre (0..99) sono esprimibili mediante 7 bit, i numeri fino a 3 cifre (100..999) mediante 10 bit, i numeri fino a 4 cifre (1000..9999) con 14 bit e via dicendo.

In generale, se noi indichiamo con $N = 10^{d} - 1$ il massimo numero intero esprimibile con d cifre decimali, in base a quanto detto prima esisterà un intero b per cui sarà valida la seguente relazione:

$$2^{b-1} \le N < 2^{b}$$

Dato però che $N = 10d - 1$ possiamo scrivere che

$$2^{b-1} \le 10^{d} - 1 < 2^{b}$$

Se eliminiamo il termine sottrattivo 1, la doppia disuguaglianza resta vera a patto di scrivere che

$$2^{b-1} < 10^{d} < 2^{b}$$

Passando ai logaritmi in base 2, possiamo inoltre scrivere che

$$b - 1 < \log_2 10^{d} < b \longrightarrow b - 1 < d \cdot \log_2 10 < b \longrightarrow \boxed{b - 1 < d \cdot 3.32 < b}$$

Dato, allora, che b è il numero di cifre binarie necessarie per rappresentare il massimo numero intero esprimibile con d cifre decimali, questa relazione dice che *b si ottiene moltiplicando d per il numero 3.32 e arrotondando il risultato all'intero superiore.* In altre parole b sarà il minimo intero maggiore del numero (3.32*d).

ADDIZIONE BINARIA

Le addizioni fondamentali tra due cifre binarie danno ovviamente luogo ai seguenti risultati:

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

SOTTRAZIONE BINARIA

Esistono due modi distinti di effettuare la sottrazione nel sistema binario: un *modo diretto*, che segue le stesse regole della sottrazione nel sistema decimale, e il *modo del complemento*, che è quello usato nei calcolatori. Rimandando a dopo l'esame del metodo del complemento, illustriamo i risultati delle sottrazioni fondamentali:

0 – 0 = 0
1 – 1 = 0
1 – 0 = 1
0 – 1 = 1 con prestito di 1 dalla cifra precedente

MOLTIPLICAZIONE BINARIA

La moltiplicazione binaria si effettua esattamente come nel sistema decimale; anzi, risulta semplificata in quanto si possono direttamente eliminare le moltiplicazioni per 0 con l'accortezza di spostare il risultato della successiva moltiplicazione di una colonna verso sinistra.

DIVISIONE BINARIA

Anche la divisione binaria avviene in modo analogo a quello nel sistema decimale, con in più il vantaggio che il rapporto tra 2 singole cifre può essere solo 0 o 1 (al contrario del sistema decimale in cui tale rapporto può andare da 0 a 9).

COMPLEMENTAZIONE

Come vedremo meglio più avanti, è possibile adottare una particolare rappresentazione dei numeri (in qualsiasi sistema di numerazione) al fine di semplificare alcune operazioni matematiche (tipicamente le sottrazioni) che coinvolgono tali numeri.

Sia b la base del sistema di numerazione nel quale dobbiamo effettuare la sottrazione (nel sistema binario b=2). Sia N un numero intero, non nullo, con K cifre nel sistema in base b. Si chiama **complemento alla base b** del numero N considerato il nuovo numero

$$N' = b^K - N$$

Questo numero N' si può calcolare molto facilmente nel seguente modo: si prendono tutte le cifre, a partire da SINISTRA, escludendo quella più a destra; di ognuna di queste cifre si effettua il complemento a b-1, ossia si calcola la differenza tra il numero (b-1) e la cifra data e la sostituisce nel numero; dell'ultima cifra a destra si effettua invece il complemento a b.

Facciamo un esempio concreto: sia dato il numero N=007426 decimale a 6 cifre; vogliamo calcolarne il complemento N' alla base 10. Applicando la semplice definizione, abbiamo che

$$N' = (10)^6 - N = 992574$$

In modo più semplice, possiamo invece applicare il metodo di complementazione delle singole cifre:

a) complemento di 0 a b-1 = 9
b) complemento di 0 a b-1 = 9
c) complemento di 7 a b-1 = 2
d) complemento di 4 a b-1 = 5
e) complemento di 2 a b-1 = 7
f) complemento di 6 a b = 4

Leggendo i risultati ottenuti dall'alto verso il basso si ottiene evidentemente il numero N' = 992574.

Questo vale dunque per il sistema decimale. Vediamo adesso come si procede nel sistema binario, dove le operazioni da seguire sono semplificate dal fatto che i simboli sono solo 2.

Sia N=10011101 il numero del quale vogliamo il **complemento a 2**; effettuando gli stessi passaggi di prima abbiamo

1 ---- 0	(complemento a 1 della cifra più a sinistra di N)
0 ---- 1	
0 ---- 1	
1 ---- 0	
1 ---- 0	
1 ---- 0	
0 ---- 1	
1 ---- 1	(complemento a 2 della cifra più a destra di N)

Deduciamo dunque che il complemento a 2 di N=10011101 è il numero N' = 01100011.

Questo è dunque il complemento a 2 di un numero binario. Esiste anche un altro tipo di complemento di un numero binario, molto facile da eseguire ed anche molto importante in quanto consente di ricavare altrettanto facilmente anche il complemento a 2. Dato un numero binario N, si definisce **complemento ad 1** di N il numero N' che si ottiene da N sostituendo gli 0 con 1 e viceversa. Per esempio, il complemento ad 1 di N=10011101 è il numero N'=01100010.

Il complemento ad 1 di N consente di ricavare immediatamente il complemento a 2 di N: infatti, se N' è il complemento ad 1 di N, il complemento a 2 di N si ottiene semplicemente sommando 1 ad N'. Ad esempio, dato sempre N=10011101, il complemento ad 1 è N'=01100010, per cui il complemento a 2 è

0	1	1	0	0	0	1	0	+	(complemento a 1 di 10011101)
							1	=	
0	1	1	0	0	0	1	1		(complemento a 2 di 10011101)

NUMERI CON SEGNO

Per **numeri segnati** intendiamo quei numeri che possono essere sia positivi sia negativi, che cioè vengono memorizzati all'interno di una memoria insieme al loro segno.

In alcuni casi, per la verità rari, il metodo di codificazione utilizzato è tale che ad ogni cifra del numero venga assegnato un numero fisso di bit: in questo modo, anche il segno del numero in questione viene considerato come un normale carattere e viene quindi trattato di conseguenza. La necessità di raggiungere spesso la massima velocità di calcolo e il conseguente uso del sistema binario hanno però suggerito un modo diverso di codificare i numeri segnati: è ormai usanza universale quella di *riservare al segno un bit alla sinistra della cifra più significativa:* Il valore di tale bit è "0" se il numero è positivo mentre è "1" se il numero è negativo.

Il modo più semplice di procedere è il seguente: il numero decimale 13 diventa 1101 nel sistema binario; allora, se si vuole memorizzare il numero +13 verrà memorizzato il numero 01101, mentre se il numero è -13, allora verrà memorizzato il numero 11101.

Ovviamente, questo è un esempio facile per spiegare il concetto; in realtà, per poter utilizzare un metodo simile è indispensabile che ai numeri da memorizzare sia riservato un spazio fisso in memoria: in primo luogo è infatti necessario poter sempre distinguere il bit del segno dagli altri bit; inoltre, quando si effettuano delle operazioni, è necessario che sia gli operandi interessati sia il risultato abbiano la stessa lunghezza.

Un fondamentale accorgimento che viene utilizzato per semplificare al massimo i circuiti logici per la realizzazione delle operazioni matematiche è il seguente: i numeri negativi non vengono memorizzati come abbiamo fatto vedere prima con il numero 13, dando cioè il valore "1" al bit del segno e poi memorizzando il valore binario esatto del numero; al contrario, si preferisce conservare il complemento (a 2 o a 1) del numero, in modo che le sottrazioni si trasformino in addizioni e quindi si possano utilizzare gli stessi circuiti. Vediamo di chiarirci meglio le idee: supponiamo che la lunghezza riservata ai numeri sia di 6 bit più il bit del segno e che si utilizzi il complemento a 2 per la memorizzazione dei numeri negativi; siano allora da memorizzare i numeri +15 e -12:

- il numero +15 è positivo (il valore binario è 001111) e quindi la sua rappresentazione in memoria sarà 0001111, dove il primo bit a sinistra è il bit del segno;
- il numero -12 è invece negativo, per cui si fa il seguente ragionamento: il bit del segno vale "1" per indicare che si tratta di un numero negativo; il valore binario del numero 12 è 001100: allora si fa il complemento a 2 di tale numero, che è 110100, e quindi si memorizza il numero 1110100. Nel momento in cui la macchina legge questi bit, dal primo deduce che si tratta di un numero negativo, per cui effettua il complemento a 2 dei successivi 6 bit e interpreta il numero nel modo corretto, ossia come -12.

Se, al posto di usare il complemento a 2, si usasse il complemento ad 1 per la memorizzazione dei numeri negativi, le cose non cambiano di molto: il valore binario del numero 12 è 001100; il

complemento ad 1 di tale valore è 110011, per cui, aggiungendo il bit del segno (posto a 1), la sequenza che viene conservata è 1110011.

OPERAZIONI CON I NUMERI SEGNATI

Una volta compreso il modo con cui vengono rappresentati e memorizzati i numeri negativi (per i numeri positivi non c'è nessuna particolare accortezza, salvo il fatto di porre a 0 il bit del segno), vediamo come si procede nell'effettuare le operazioni con i numeri affetti da segno.

Per comprendere i vari procedimenti, cominciamo col considerare numeri privi di segno ed a lunghezza fissa.

Siano A e B due numeri, in un generico sistema di numerazione, privi di segno, con K cifre decimali e tali che A≥B. Con questi numeri, possiamo effettuare le seguenti operazioni fondamentali:

$$a)A+B \qquad b)A-B \qquad c)-A+B \qquad d)-A-B$$

La prima operazione non richiede nessuna osservazione, dato che si tratta di una semplice somma tra due numeri positivi.

La seconda operazione consiste invece nel sottrarre il numero B dal numero A. Vogliamo far vedere che questa sottrazione si può in realtà effettuare tramite una addizione, a patto di prendere gli opportuni accorgimenti. Analiticamente, se indichiamo con B' il complemento a 2 di B, basta fare i seguenti passaggi:

$$A+B'=A+(b^K-B)=A-B+b^K$$

In base a questa relazione, se noi sommiamo A ed il complemento a 2 di B, otteniamo, come risultato, la quantità A-B che ci interessa più un termine b^K. Questo termine non è altro che un riporto unitario sulla prima cifra a sinistra del risultato: possiamo perciò concludere che l'operazione A-B diventa l'operazione A+B' se, nella somma risultante, si trascura l'ultimo riporto, quello cioè in posizione K+1.

Facciamo un esempio con 2 numeri decimali: siano A = 43517 e B = 26106 i due numeri. Vogliamo calcolare A-B con il procedimento appena descritto. Effettuiamo, allora, per prima cosa, il complemento a 10 di B:

$$B' = (10)6 - 26106 = 73894$$

Abbiamo allora che

$$\begin{array}{cccccc} 4 & 3 & 5 & 1 & 7 & + \\ 7 & 3 & 8 & 9 & 4 & = \\ \hline \underset{=}{1} & 1 & 7 & 4 & 1 & 1 \end{array}$$

Abbiamo ottenuto un riporto nella 1° posizione a sinistra: eliminando tale riporto, otteniamo 17411, che è proprio la differenza tra 43517 e 26106.

Passiamo ad un esempio nel sistema binario: vogliamo la differenza tra i numeri $A=010111$ e $B=001100$. Cominciamo con l'effettuare il complemento a 2 di B: facendo prima il complemento ad 1 e poi sommando 1, otteniamo

$$B=001100 \xrightarrow{\text{complemento ad 1}} 110011 \xrightarrow{\text{complemento a 2}} B'=110100$$

Allora

$$\begin{array}{cccccccc} & 0 & 1 & 0 & 1 & 1 & 1 & + \\ & 1 & 1 & 0 & 1 & 0 & 0 & = \\ \hline \underline{1} & 0 & 0 & 1 & 0 & 1 & 1 & \end{array}$$

Anche in questo caso, abbiamo ottenuto un riporto unitario nella 1° posizione a sinistra, per cui, eliminando tale riporto, otteniamo il risultato corretto, ossia 001011. Che si tratti del risultato corretto si può verificare convertendo in decimale i numeri coinvolti: infatti, dato che A=23 e B=12, la differenza è 11, che è proprio il risultato trovato.

Adesso consideriamo l'altra operazione, ossia -A+B, che darà ovviamente un risultato negativo in quanto A>B. In questo caso, se effettuiamo la somma di B con il complemento a 2 di A, otteniamo direttamente (senza alcun riporto) il complemento a 2 del valore cercato (cui ovviamente va aggiunto il segno negativo), come si deduce dai seguenti passaggi:

$$A'+B=(b^K-A)+B=b^K+(-A+B)=b^K-(A-B)=(A-B)'$$

Sommando A'+B, otteniamo il complemento a 2 di A-B, che rappresenta il valore assoluto del risultato da noi cercato.

Facciamo anche in questo caso un esempio, direttamente nel sistema binario. Consideriamo i numeri $A=010111$ (23 in decimale) e $B=001100$ (12 in decimale). Vogliamo calcolare -A+B. Per prima cosa, dobbiamo effettuare il complemento a 2 di A:

$$A=010111 \xrightarrow{\text{complemento ad 1}} 101000 \xrightarrow{\text{complemento a 2}} A'=101001$$

Adesso dobbiamo sommare A'+B:

$$\begin{array}{cccccccc} & 1 & 0 & 1 & 0 & 0 & 1 & + \\ & 0 & 0 & 1 & 1 & 0 & 0 & = \\ \hline // & 1 & 1 & 0 & 1 & 0 & 1 & \end{array}$$

Non abbiamo ottenuto alcun riporto e il numero 110101 ottenuto non è altro che il complemento a 2 del risultato cercato, che quindi è

$$110101 \xrightarrow{\text{complemento ad 1}} 001010 \xrightarrow{\text{complemento a 2}} 001011 \quad (11 \text{ in decimale})$$

Infine, l'ultima operazione da considerare è -A-B, che darà ovviamente anch'essa risultato negativo: con ragionamenti analoghi ai precedenti, abbiamo che

$$A'+B'=(b^K-A)+(b^K-B)=2b^K-(A+B)=b^K+\left[b^K-(A+B)\right]=b^K+(A+B)'$$

In questo caso, quindi, si ottiene un riporto sulla cifra più a sinistra: trascurando tale riporto, rimane (A+B)', ossia il complemento a 2 del risultato cercato (cui va ovviamente aggiunto il segno negativo).

Facciamo un esempio con gli stessi numeri $A=010111$ (23 in decimale) e $B=001100$ (12 in decimale) dei casi precedenti. Vogliamo calcolare -A-B, per cui dobbiamo per prima cosa effettuare il complemento a 2 sia di A sia di B:

$$A=010111 \xrightarrow{\text{complemento ad 1}} 101000 \xrightarrow{\text{complemento a 2}} A'=101001$$
$$B=001100 \xrightarrow{\text{complemento ad 1}} 110011 \xrightarrow{\text{complemento a 2}} B'=110100$$

Adesso dobbiamo sommare A'+B':

$$\begin{array}{cccccccc} & 1 & 0 & 1 & 0 & 0 & 1 & + \\ & 1 & 1 & 0 & 1 & 0 & 0 & = \\ \hline \underline{1} & 0 & 1 & 1 & 1 & 0 & 1 & \end{array}$$

Trascurando il riporto, il numero 011101 rappresenta il complemento a 2 del risultato cercato, che quindi è

$$011101 \xrightarrow{\text{complemento ad 1}} 100010 \xrightarrow{\text{complemento a 2}} 100011 \quad (35 \text{ in decimale})$$

Passiamo adesso a considerare i ***casi di interesse pratico in cui i numeri considerati nelle operazioni presentano un bit che ne indica il segno.*** Ciò significa, dunque, fondamentalmente tre cose: che il numero di bit riservati a ciascun numero è fisso; che il bit più a sinistra è quello del segno; che i numeri negativi sono rappresentati mediante il proprio complemento a 2 (dove la complementazione coinvolte ovviamente anche il bit del segno[1]). In particolare, supponiamo che la lunghezza di ciascun numero sia di n+1 bit (incluso quello del segno).

Consideriamo dunque due numeri A e B interi positivi e tali che A>B, ciascuno della lunghezza di n+1 bit incluso il segno; le situazioni possibili sono le stesse esaminate prima:

$$a) A+B \qquad b) A-B \qquad c) -A+B \qquad d) -A-B$$

I risultati delle 4 operazioni, effettuati con il metodo del complemento a 2, sono gli stessi descritti prima, con in più il fatto di considerare, nelle operazioni, anche il bit del segno. Vediamo allora di chiarire quanto detto con degli esempi concreti.

Per quanto riguarda la somma, non ci sono osservazioni particolari: basta effettuare la somma dei singoli bit per ottenere il risultato corretto (che risulta priva di riporto).

Consideriamo allora l'operazione A-B: in modo analogo a quanto visto prima, abbiamo che

$$A+B' = A+(b^K - B) = A-B+b^K$$

In base a questa relazione, se noi sommiamo A ed il complemento a 2 di B, otteniamo, come risultato, la quantità A-B che ci interessa più un termine b^K. Trascurando tale termine (ossia il riporto a sinistra del bit del segno), otteniamo il risultato voluto.

Consideriamo ad esempio i numeri A=23 e B=12. Vogliamo calcolare A-B con il procedimento appena descritto, per cui dobbiamo rappresentare A come 0.010111 (dove il "punto" separa il bit del segno dagli altri bit) e B con il suo complemento a 2 (incluso il segno):

$$B = 0.001100 \xrightarrow{\text{complemento ad 1}} 1.110011 \xrightarrow{\text{complemento a 2}} B' = 1.110100$$

Adesso effettuiamo la somma:

$$\begin{array}{cccccccc} & 0. & 0 & 1 & 0 & 1 & 1 & 1 & + \\ & 1. & 1 & 1 & 0 & 1 & 0 & 0 & = \\ \hline \underline{1} & 0. & 0 & 0 & 1 & 0 & 1 & 1 & \end{array}$$

[1] Questo significa che, se il numero interessato è -12 e i bit considerati sono 5 (incluso il segno), la rappresentazione binaria è la seguente: per prima cosa, si considera 12 in binario, ossia 1100; poi si aggiunge il bit del segno a 0, ottenendo 01100; quindi si effettua il complemento a 1, che vale 10011; infine si effettua il complemento a 2, ottenendo 10100.

Abbiamo ottenuto un riporto a sinistra del bit del segno: eliminando tale riporto, otteniamo il numero 0.001011, che rappresenta il risultato corretto, ossia +11 in decimale.

Passiamo adesso all'operazione -A+B, che darà ovviamente risultato negativo in quanto A>B. In questo caso, se effettuiamo la somma di B con il complemento a 2 di A, otteniamo direttamente (senza alcun riporto) il complemento a 2 del valore cercato (che ovviamente conterrà il bit del segno ad 1), come si deduce dai seguenti passaggi:

$$A'+B=(b^K-A)+B=b^K+(-A+B)=b^K-(A-B)=(A-B)'$$

Consideriamo ad esempio i numeri $A=0.010111$ (23 in decimale) e $B=0.001100$ (12 in decimale), dove il "punto" separa il bit del segno dagli altri bit. Vogliamo calcolare -A+B. Per prima cosa, dobbiamo effettuare il complemento a 2 di A, includendo il bit del segno:

$$A=0.010111 \xrightarrow{\text{complemento ad 1}} 1.101000 \xrightarrow{\text{complemento a 2}} A'=1.101001$$

Adesso dobbiamo sommare A'+B:

	1.	1	0	1	0	0	1	+
	0.	0	0	1	1	0	0	=
//	1.	1	1	0	1	0	1	

Non abbiamo ottenuto alcun riporto e il numero 1.110101 ottenuto non è altro che il complemento a 2 del risultato cercato, che quindi è

$$1.110101 \xrightarrow{\text{complemento ad 1}} -001010 \xrightarrow{\text{complemento a 2}} -001011 \quad (\text{-11 in decimale})$$

In pratica, quindi, ***una volta ottenuto il risultato dell'operazione, il bit del segno serve a capire se il numero è negativo o meno: i restanti bit forniscono il valore del numero se il numero stesso è positivo, mentre ne indicano il complemento a 2 se è negativo.***

Infine, l'ultima operazione da considerare è -A-B, che darà ovviamente anch'essa risultato negativo: con ragionamenti analoghi ai precedenti, abbiamo che

$$A'+B'=(b^K-A)+(b^K-B)=2b^K-(A+B)=b^K+\left[b^K-(A+B)\right]=b^K+(A+B)'$$

In questo caso, quindi, si ottiene un riporto a sinistra del bit del segno: trascurando tale riporto, rimane (A+B)', ossia il complemento a 2 del risultato cercato (che ovviamente presenterà il bit del segno ad 1).

Facciamo un esempio con gli stessi numeri $A=010111$ (23 in decimale) e $B=001100$ (12 in decimale) dei casi precedenti. Vogliamo calcolare -A-B, per cui dobbiamo per prima cosa effettuare il complemento a 2 sia di A sia di B:

$$A=0.010111 \xrightarrow{\text{complemento ad 1}} 1.101000 \xrightarrow{\text{complemento a 2}} A'=1.101001$$
$$B=0.001100 \xrightarrow{\text{complemento ad 1}} 1.110011 \xrightarrow{\text{complemento a 2}} B'=1.110100$$

Adesso dobbiamo sommare A’+B’:

$$\begin{array}{cccccccc} & 1. & 1 & 0 & 1 & 0 & 0 & 1 & + \\ & 1. & 1 & 1 & 0 & 1 & 0 & 0 & = \\ \hline \underline{1} & 1. & 0 & 1 & 1 & 1 & 0 & 1 & \end{array}$$

Trascurando il riporto a sinistra del bit del segno, il numero 1.011101 così ottenuto rappresenta il complemento a 2 del risultato cercato, che quindi è

$$1.011101 \xrightarrow{\text{complemento ad 1}} -100010 \xrightarrow{\text{complemento a 2}} -100011 \quad (-35 \text{ in decimale})$$

Osservazione: uso del complemento ad 1

Non tutti i processori usano la rappresentazione dei numeri negativi con il complemento a 2, ma ce ne sono alcuni che usano il complemento ad 1. In questo caso, i discorsi appena conclusi sulle operazioni con numeri affetti da segno non cambiano salvo che in un accorgimento fondamentale: mentre, con il complemento a 2, l’eventuale riporto a sinistra del bit del segno va semplicemente eliminato, con il complemento a 1 esso va sommato in coda al risultato. Consideriamo degli esempi per capirci meglio.

Consideriamo i numeri A=23 e B=12. Vogliamo calcolare A-B. La codifica binaria di A, incluso il bit del segno, è 0.010111, mentre quella di B, usando il complemento ad 1, è 1.110011. Effettuiamo allora la somma A+B’:

$$\begin{array}{cccccccc} & 0. & 0 & 1 & 0 & 1 & 1 & 1 & + \\ & 1. & 1 & 1 & 0 & 0 & 1 & 1 & = \\ \hline \underline{1} & 0. & 0 & 0 & 1 & 0 & 1 & 0 & \end{array}$$

Abbiamo ottenuto un riporto a sinistra del bit del segno; tale riporto va sommato in coda:

$$\begin{array}{cccccccc} 0. & 0 & 0 & 1 & 0 & 1 & 0 & + \\ & & & & & & 1 & = \\ \hline 0. & 0 & 0 & 1 & 0 & 1 & 1 & \end{array}$$

Il numero così ottenuto è il risultato voluto: essendo il bit del segno a 0, deduciamo che il numero è positivo ed il suo valore numero è quello indicato dai restanti bit, ossia +11.

Adesso consideriamo gli stessi due numeri A e B e calcoliamo -A+B: la codifica binaria di A deve essere attraverso il suo complemento ad 1, per cui è 1.101000, mentre quella di B è semplicemente 0.001100. Effettuiamo allora la somma A’+B:

$$\begin{array}{cccccccc} & 1. & 1 & 0 & 1 & 0 & 0 & 0 & + \\ & 0. & 0 & 0 & 1 & 1 & 0 & 0 & = \\ \hline \underline{0} & 1. & 1 & 1 & 0 & 1 & 0 & 0 & \end{array}$$

In questo caso, non c’è nessun riporto da sommare in coda. Il numero così ottenuto è il risultato voluto: essendo il bit del segno a 0, deduciamo che il numero è positivo ed il suo valore numero è quello indicato dai restanti bit, ossia +11.

Infine, consideriamo -A-B, per cui dobbiamo sommare A'=1.101000 e B'=1.110011:

	1.	1	0	1	0	0	0 +
	1.	1	1	0	0	1	1 =
1	1.	0	1	1	0	1	1

C'è questa volta un riporto, che va quindi sommato in coda:

	1.	0	1	1	0	1	1 +
							1 =
	1.	0	1	1	1	0	0

Il numero così ottenuto è il risultato voluto: essendo ad 1 il bit del segno, deduciamo che si tratta di un numero negativo il cui valore assoluto è dato dal complemento ad 1 dei restanti bit; facendo quindi tale complemento ad 1, si ottiene 100011, ossia 33, da cui concludiamo che il risultato dell'operazione è -33.

Il fenomeno dell'overflow

Per concludere facciamo una importante osservazione: quando si opera con dati a lunghezza fissa, ossia con dati per i quali è riservati un numero fisso e immutabile di bit, spesso si ha a che fare con il fenomeno del cosiddetto **overflow** (traboccamento), per cui il risultato di una operazione risulta avere una lunghezza maggiore di quella riservata agli addendi. Un esempio è il seguente: siano da sommare i numeri +51 e +41, le cui rappresentazioni binarie, incluso il segno, sono rispettivamente 0.110011 e 0.101001; eseguendo la somma si ottiene

$$0.110011 + 0.101001 = 1.011100$$

Come risultato della somma di due numeri positivi abbiamo ottenuto un numero negativo e ciò è ovviamente sbagliato: in questi casi, il calcolatore riconosce il verificarsi del fenomeno e tronca il risultato nel suo bit più significativo, fornendo come risultato 0.011100, cioè il valore corretto.

DIFETTI DEL METODO DELLA COMPLEMENTAZIONE

Il metodo della complementazione ha dunque il grande pregio di ricondurre le operazioni di sottrazione ad operazioni di addizione, consentendo perciò di usare un minor numero di circuiti. D'altra parte, essendo presenta due difetti rilevanti: il primo è che, per le operazioni di prodotto e di quoziente, non è altrettanto efficace; il secondo è che comporta uno *spreco di risorse hardware*.

Per capire cosa intendiamo per "spreco di hardware", facciamo il seguente ragionamento: supponiamo di avere a disposizione 5 bit per rappresentare un numero; a tali 5bit corrispondono 32 possibili combinazioni, per cui si possono teoricamente rappresentare 32 numeri. Con il metodo della complementazione, invece, sacrificando il primo bit per il segno, si possono rappresentare solo 15 numeri, come evidenziato nella tabelle seguenti (i numeri negativi sono rappresentati a sinistra nel complemento ad 1 e a destra nel complemento a 2):

+7	0.	0	1	1	1
+6	0.	0	1	1	0
+5	0.	0	1	0	1
+4	0.	0	1	0	0
+3	0.	0	0	1	1
+2	0.	0	0	1	0
+1	0.	0	0	0	1
+0	0.	0	0	0	0
−0	1.	1	1	1	1
−1	1.	1	1	1	0
−2	1.	1	1	0	1
−3	1.	1	1	0	0
−4	1.	1	0	1	1
−5	1.	1	0	1	0
−6	1.	1	0	0	1
−7	1.	1	0	0	0

+7	0.	0	1	1	1
+6	0.	0	1	1	0
+5	0.	0	1	0	1
+4	0.	0	1	0	0
+3	0.	0	0	1	1
+2	0.	0	0	1	0
+1	0.	0	0	0	1
0	0.	0	0	0	0
−1	1.	1	1	1	1
−2	1.	1	1	1	0
−3	1.	1	1	0	1
−4	1.	1	1	0	0
−5	1.	1	0	1	1
−6	1.	1	0	1	0
−7	1.	1	0	0	1

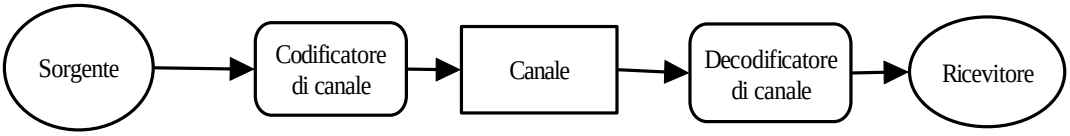
Quindi, mentre senza la complementazione potremmo rappresentare i numeri da -16 a +15, con la complementazione (sia ad 1 sia a 2) possiamo rappresentare i numeri solo da -7 a +7.

Una cosa interessante da osservare, nelle tabella appena riportate, è che ***lo zero può essere rappresentato in due diversi modi con il complemento ad 1 e in un modo solo con il complemento a 2.*** In realtà, usando il complemento ad 1, la rappresentazione usata per lo zero è 1.1111: il motivo è che, dato un qualsiasi numero binario, se lo si somma al suo complemento si ottiene proprio una sequenza di 1, incluso il segno.

Codici binari

INTRODUZIONE

Consideriamo lo schema semplificato di un generico schema di trasmissione tra sorgente e ricevitore:

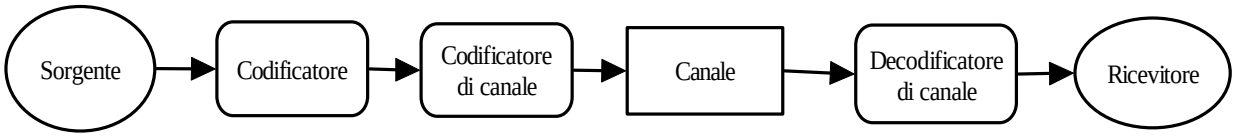


La “*sorgente*” genera il segnale (quale che sia la sua natura) da trasmettere al ricevitore (cioè all’utente); tale segnale non viene però trasmesso così com’è, ma viene in qualche modo codificato da un opportuno dispositivo, che prende appunto il nome di “*codificatore di canale*”; il segnale codificato emesso dal codificatore viene inviato al canale di trasmissione, il quale rappresenta tutti quei dispositivi necessari per la trasmissione del segnale stesso; in particolare, compito del canale è

quello di far arrivare il segnale codificato ad un altro dispositivo, il "*decodificatore di canale*", il quale, comportandosi in modo inverso al codificatore, ricostruisce il segnale così come è stato emesso dalla sorgente e lo invia al "*ricevitore*".

La necessità di codificare il segnale emesso dalla sorgente deriva da vari fattori, primo tra i quali i problemi legati al funzionamento del canale: infatti, trattandosi di un insieme di cavi e dispositivi fisici, abbiamo detto che è possibile che esso commetta degli "*errori*" nella trasmissione del segnale. Allora, il "codificatore di canale" viene progettato in modo che la sequenza di bit che corrisponde al segnale inviato dalla sorgente sia tale da permettere al decodificatore di canale di rilevare ed eventualmente correggere gli errori verificatisi durante la trasmissione.

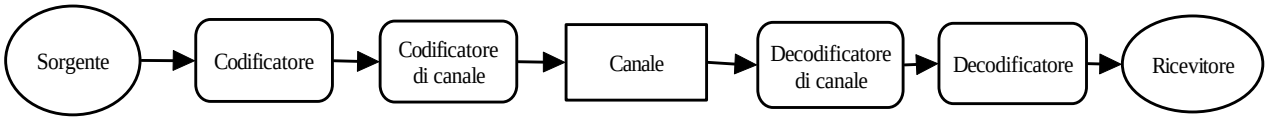
Per quanto riguarda il segnale che viene trasmesso dal canale, ci occupiamo in questa sede di segnali digitali, costituiti cioè da due soli livelli (alto e basso). Dobbiamo allora perfezionare quello schema aggiungendo un ulteriore componente: si tratta del cosiddetto "codificatore", ossia di un dispositivo che, dato il segnale emesso dalla sorgente, vi associa una corrispondente sequenza di bit (che costituisce appunto la codifica del segnale) necessaria per la trasmissione del segnale stesso.



E' opportuno sottolineare le differenze di comportamento tra il "codificatore" ed il "codificatore di canale": il primo riceve in ingresso il segnale emesso dalla sorgente e genera in uscita la sequenza di bit che serve per la trasmissione di tale segnale; il secondo, invece, riceve ingresso la sequenza di bit emessa dal primo e opera su di essa quelle manipolazioni che permetteranno al decodificatore di canale di recuperare eventuali errori dovuti alla trasmissione tramite il canale.

Ovviamente, così come il decodificatore di canale si comporta in modo pressoché inverso al codificatore di canale, ci dovrà essere un ulteriore dispositivo che si comporta in modo inverso al codificatore: tale dispositivo riceve in ingresso la sequenza di bit ripulita, tramite il decodificatore di canale, degli errori dovuti alla trasmissione e, da questa sequenza, ricostruisce il segnale emesso dalla sorgente inviandolo al ricevitore.

Possiamo perciò concludere che lo schema più o meno completo di un sistema di trasmissione è il seguente:



Nei prossimi paragrafi ci occupiamo di analizzare sia i codici binari utilizzati per la codifica dei simboli sia i codici utilizzati per la rilevazione e correzione degli errori di trasmissione.

GENERALITÀ SUI CODICI BINARI

Abbiamo detto che il codificatore ha il compito di associare, alla sequenza di simboli emessi dalla sorgente, una opportuna sequenza di bit. E' ovvio che questa sequenza di bit deve essere tale che, una volta arrivata al decodificatore, quest'ultimo sia in grado di ricostruire esattamente, a partire da essa, il segnale emesso dalla sorgente. Questo è possibile solo se il codice binario utilizzato gode della proprietà di essere "**univocamente decodificabile**": deve cioè essere tale che la decodifica possa essere una ed una sola, in modo da avere la garanzia che il ricevitore riceva ciò che la sorgente ha emesso.

E' ovvio che i codici ai quali si può pensare sono infiniti. In linea di massima, i requisiti che un codice deve avere, oltre appunto alla "univoca decodificabilità", sono i seguenti:

- deve prevedere il più basso numero di bit possibile, in modo da ridurre i tempi di trasmissione e di elaborazione e da semplificare i circuiti preposti a questi operazioni;

- deve inoltre consentire di "risparmiare in banda", ossia occupare nella minima misura possibile la banda passante del canale.

A requisiti di questo genere rispondono sia i codici "a lunghezza fissa" sia quelli "a lunghezza variabile". Che cosa si intende con queste espressioni? Un **codice a lunghezza fissa** è un codice in cui ad ogni simbolo viene associato sempre lo stesso numero di bit: un esempio tipico è la codifica ASCII dei caratteri. Un **codice a lunghezza variabile** è invece evidentemente un codice in cui il numero di bit associati a ciascun simbolo è variabile. Nei seguenti paragrafi ci occupiamo solo dei codici a lunghezza fissa.

Supponiamo adesso che la nostra sorgente abbia un **alfabeto** costituito da un numero finito di **simboli** (sia essi cifre decimali o lettere o simboli di qualsiasi tipo): a ciascuno di tali simboli il codificatore associa una sequenza di bit che prende il nome di **parola di codice**.

Un importante concetto è quello di *distanza* tra due parole di codice: date le due parole, si definisce **distanza** tra di esse il numero di bit che bisogna cambiare in ognuna per arrivare all'altra. Per esempio, per un codice a 4 bit, se consideriamo le parole 1010 e 0101, è ovvio che la distanza è 4, in quanto le due parole hanno bit tutti diversi. Se invece le due parole fossero 1010 e 1011, la loro distanza sarebbe 1, visto che solo l'ultimo bit è diverso.

Se l'alfabeto della sorgente comprende N simboli, il codice da progettare deve comprendere parole da n bit, dove n deve ovviamente soddisfare alla relazione $2^n \geq N$: se non dovesse valere il segno di uguaglianza, per cui il codice presenta più di N combinazioni, le combinazioni in più rappresentano la cosiddetta **ridondanza**. Vedremo in seguito che proprio la ridondanza consente di ideare codici per la rilevazione ed eventualmente la correzione degli errori.

E' ovvio che *il numero n di bit di ciascuna parola del codice non individua in modo univoco il codice stesso:* serve anche definire una corrispondenza biunivoca tra ciascun simbolo dell'alfabeto sorgente e ciascuna parola del codice.

CODICI NUMERICI

A partire da questo paragrafo ci occupiamo nel dettaglio dei codici usati per rappresentare le 10 cifre della numerazione decimale: a tali codici si dà il nome di **codici numerici**. Vale subito un risultato fondamentale: dovendo rappresentare 10 simboli (cioè appunto le 10 cifre decimali), abbiamo bisogno di un codice di almeno 4 bit, per cui tutti i codici numerici che esamineremo presentano parole da 4 bit.

Il primo codice numerico ad essere introdotto fu il *codice naturale*, più noto come **codice BCD** (dove BCD sta per *Binary Coded Decimal*):

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

Il codice BCD è un **codice pesato**, in quanto ciascun bit ha un **peso** che dipende dalla posizione occupata nella parola:

$$\underset{\text{posizione}}{\text{peso}} \quad \underset{3}{2^3} \quad \underset{2}{2^2} \quad \underset{1}{2^1} \quad \underset{0}{2^0}$$

Partendo (sempre) da destra, il bit in posizione 0 ha peso 1, il bit in posizione 1 ha peso 2, il bit in posizione 2 ha peso 4 ed il bit in posizione 3 ha peso 8. Per questo motivo, si parla anche di **codice 8421**, proprio per indicare i pesi dei vari bit.

Un'altra osservazione importante è quella per cui il codice BCD è un esempio di **codice a ridondanza**: infatti, delle 2^4=16 combinazioni possibili con 4 bit, il codice ne sfrutta solo 10, mentre le rimanenti 6 costituiscono appunto la ridondanza.

Un altro esempio di codice pesato è il **codice 2421**, detto anche *codice AIKEN*: in questo caso, come si osserva dal nome, i pesi non sono più in ordine crescente verso sinistra come nel codice 8421 ed inoltre ci sono due bit aventi lo stesso peso. Il codice è il seguente:

$$\begin{aligned}
0 &\rightarrow 0\ 0\ 0\ 0 \\
1 &\rightarrow 0\ 0\ 0\ 1 \\
2 &\rightarrow \begin{cases} 1\ 0\ 0\ 0 \\ 0\ 0\ 1\ 0 \end{cases} \\
3 &\rightarrow \begin{cases} 1\ 0\ 0\ 1 \\ 0\ 0\ 1\ 1 \end{cases} \\
4 &\rightarrow \begin{cases} 0\ 1\ 0\ 0 \\ 1\ 0\ 1\ 0 \end{cases} \\
5 &\rightarrow \begin{cases} 1\ 0\ 1\ 1 \\ 0\ 1\ 0\ 1 \end{cases} \\
6 &\rightarrow \begin{cases} 1\ 1\ 0\ 0 \\ 0\ 1\ 1\ 0 \end{cases} \\
7 &\rightarrow \begin{cases} 1\ 1\ 0\ 1 \\ 0\ 1\ 1\ 1 \end{cases} \\
8 &\rightarrow 1\ 1\ 1\ 0 \\
9 &\rightarrow 1\ 1\ 1\ 1
\end{aligned}$$

Questo codice presenta due sostanziali differenze con il codice BCD: da un lato, esso non presenta ridondanza, in quanto tutte e 16 le possibili combinazioni vengono utilizzate; dall'altro, però, dato che ci sono alcune cifre decimali (da 2 a 7) che ammettono due distinte rappresentazioni, è ovvio che non c'è corrispondenza biunivoca tra i simboli dell'alfabeto e le parole di codice.

Ovviamente, una volta che si decide di usare questo codice, bisogna stabilire, per quelle cifre che ammettono due distinte codifiche, quale codifica utilizzare. Teoricamente, si potrebbe scegliere arbitrariamente, per ciascuna cifra, quale tra le due possibili codifiche utilizzare. In realtà, si usa un altro criterio: consideriamo per esempio le cifre decimali 2 e 7 (una è il complemento a 9 dell'altra): il 2 si può rappresentare con 1000 o con 0010, mentre il 7 con 0111 oppure con 1101. Si osserva che la parola 1000 è il complemento ad 1 della parola 0111, così come 0010 è il complemento ad 1 di 1101.

La scelta viene allora fatta nel modo seguente: se per la cifra 2 si sceglie 1000, per la cifra 7 si dovrà necessariamente utilizzare la parola 0111, ossia il complemento ad 1 di quella scelta per il 2.

In modo del tutto analogo si procede per le altre coppie di cifre decimali che sono una il complemento a 9 dell'altra: ad esempio, se per la cifra 3 si sceglie la parola 1001, per la cifra 6 (complemento a 9 di 3) si dovrà usare 0110.

Codici che godono di questa proprietà sono definiti **codici ad autocomplementazione**. Il codice 2421 appena esaminato è un codice ad autocomplementazione ed anche un codice pesato. Esistono però anche codici ad autocomplementazione non pesati.

Il requisito necessario perché un codice pesato sia ad autocomplementazione è che la somma dei pesi sia pari a 9. A questo requisito risponde ad esempio il **codice 5211**, le cui parole sono le seguenti:

$$0 \rightarrow 0\ 0\ 0\ 0 \qquad 9 \rightarrow 1\ 1\ 1\ 1$$
$$1 \rightarrow \begin{cases} 0\ 0\ 0\ 1 \\ 0\ 0\ 1\ 0 \end{cases} \qquad 8 \rightarrow \begin{cases} 1\ 1\ 1\ 0 \\ 1\ 1\ 0\ 1 \end{cases}$$
$$2 \rightarrow \begin{cases} 0\ 1\ 0\ 0 \\ 0\ 0\ 1\ 1 \end{cases} \qquad 7 \rightarrow \begin{cases} 1\ 0\ 1\ 1 \\ 1\ 1\ 0\ 0 \end{cases}$$
$$3 \rightarrow \begin{cases} 0\ 1\ 0\ 1 \\ 0\ 1\ 1\ 0 \end{cases} \qquad 6 \rightarrow \begin{cases} 1\ 0\ 1\ 0 \\ 1\ 0\ 0\ 1 \end{cases}$$
$$4 \rightarrow 0\ 1\ 1\ 1 \qquad 5 \rightarrow 1\ 0\ 0\ 0$$

A proposito dei codici pesati, è possibile dimostrare il seguente risultato: *affinché un codice numerico sia pesato, è necessario scegliere i pesi in modo che la loro somma sia compresa tra 9 e 15.* Ovviamente, questo non basta per definire un codice pesato: infatti, i pesi vanno anche scelti in modo tale che il codice possa rappresentare tutte le cifre decimali. Per esempio, un codice numerico pesato che soddisfa questi requisiti è il **codice 7421**: la somma dei pesi è 14 ed è facile verificare come esso consenta di rappresentare tutte le cifre decimali.

Sempre riguardo i codici pesati, abbiamo finora utilizzato sempre pesi positivi: la conseguenza fondamentale di ciò è che la cifra decimale 0 poteva essere rappresentata solo dalla combinazione 0000. Le cose cambiano, invece, se si prendono uno o più pesi negativi. Per esempio, un codice pesato ad autocomplementazione avente un peso negativo è il **codice 631-1**, dove il bit in posizione zero ha evidentemente peso -1:

$$0 \rightarrow \begin{cases} 0\ 0\ 0\ 0 \\ 0\ 0\ 1\ 1 \end{cases} \qquad 9 \rightarrow \begin{cases} 1\ 1\ 0\ 0 \\ 1\ 1\ 1\ 1 \end{cases}$$
$$1 \rightarrow 0\ 0\ 1\ 0 \qquad 8 \rightarrow 1\ 1\ 0\ 1$$
$$2 \rightarrow 0\ 1\ 0\ 1 \qquad 7 \rightarrow 1\ 0\ 1\ 0$$
$$3 \rightarrow 0\ 1\ 0\ 0 \qquad 6 \rightarrow 1\ 0\ 1\ 1$$
$$4 \rightarrow 0\ 1\ 1\ 0 \qquad 5 \rightarrow 1\ 0\ 0\ 1$$

Un altro codice pesato con pesi negativi è il **codice 732-1**, anche se non si tratta di un codice ad autocomplementazione.

Codice ad accesso 3

Oltre ai codici numeri pesati, ce ne sono alcuni non pesati altrettanto importanti. Un esempio classico è il cosiddetto **codice ad eccesso 3**: le parole di tale codice si ottengono semplicemente da quelle del codice BCD sommando la cifra decimale 3 (cioè 0011 in binario):

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

Rispetto al codice BCD, questo è dunque un codice non pesato, ma ad autocomplementazione. Si tratta di un codice molto importante in quanto ci sono alcuni **circuiti contatori** che lo usano.

Codici a rilevazione e correzione di errore

INTRODUZIONE

Supponiamo di dover trasmettere un segnale binario, ossia una sequenza di bit, da una certa sorgente ad un certo ricevitore. Possiamo schematizzare l'apparato di tramissione nel modo seguente:

La "*sorgente numerica*" è quella che genera il segnale binario da trasmettere, ossia una sequenza di simboli 1 e 0; il "*canale binario*" rappresenta invece tutti quei dispositivi necessari per la trasmissione del segnale dalla sorgente al "*ricevitore*".

I problemi, in uno schema di tramissione di questo tipo, vengono proprio dal canale binario: infatti, trattandosi di un insieme di cavi e disposiviti fisici, è possibile che esso commetta degli "*errori*" nella tramissione del segnale: in parole povere, è possibile che, nonostante la sorgente abbia emesso il simbolo 1, il ricevitore riceva il simbolo 0 e viceversa.

N.B. Il problema degli errori sui bit non è presente solo nei sistemi di trasmissione a distanza, ma anche, per esempio, nei calcolatori, le cui memorie sono talvolta soggette ad errori dovuti a picchi di voltaggio sulla linea elettrica e ad altre cause.

E' importante allora studiare, da un punto di vista probabilistico, la possibilità che le informazioni che arrivano al ricevitore siano diverse da quelle emesse dalla sorgente.

A tal fine, è possibile caratterizzare il canale binario mediante dei concetti probabilistici; in particolare, è evidente che il canale commette un errore in due soli casi:

- il primo caso è che, in corrispondenza della tramissione di un 1, esso faccia ricevere uno 0;
- il secondo caso, invece, è ovviamente che, in corrispondenza della tramissione di uno 0, esso faccia ricevere un 1.

Possiamo associare a questi due casi due eventi:

evento A : 1 ricevuto in corrispondenza di uno 0 trasmesso

evento B : 0 ricevuto in corrispondenza di un 1 trasmesso

Ancora più sinteticamente, possiamo usare il concetto di probabilità condizionate e scrivere che

$$\text{evento A : } 1R|0T$$

$$\text{evento B : } 0R|1T$$

Prendono allora il nome di "**probabilità di transizione**" la probabilità che si verifichi l'evento A e la probabilità che si verifichi l'evento B:

$$\text{probabilità di transizione : } \begin{cases} p_0 = P(1R|0T) \\ p_1 = P(0R|1T) \end{cases}$$

Chiaramente, quindi, p_0 indica la probabilità che sia ricevuto un 1 quando invece la sorgente ha emesso uno 0, mentre p_1 indica la probabilità che sia ricevuto uno 0 quando invece la sorgente ha emesso un 1.

Si dice che il canale binario considerato è "**simmetrico**" quando $p_0=p_1$: ciò significa dire che la probabilità di sbagliare simbolo è identica sia quando viene trasmetto uno 0 sia quando viene trasmesso un 1.

Nel seguito, assumeremo sempre valida questa ipotesi: indichiamo perciò iin generale con $p=p_0=p_1$ la probabilità che il canale trasmetta al ricevitore un simbolo diverso da quello che è stato emesso dalla sorgente.

A questo punto, quando si verifica un "errore"? L'evento "errore", che indichiamo con E, si verifica quando il bit ricevuto è diverso dal bit trasmesso. Vogliamo P(E). Utilizzando il noto teorema delle probabilità totali, si dimostra quanto segue: *per ogni simbolo emesso dalla sorgente, la probabilità che si verifichi un errore, ossia la probabilità che il ricevitore riceva il simbolo sbagliato, è pari proprio a p, cioè alla probabilità che sia il canale a sbagliare il simbolo.* Questo, ovviamente, nell'ipotesi di canale simmetrico.

Dimostriamo il risultato appena enunciato. Per farlo, ossia per calcolare P(E), ricordiamo l'enunciato del **teorema delle probabilità totali**, del quale ripetiamo perciò l'enunciato:

Teorema *- Sia dato lo spazio dei campioni S relativo ad un certo fenomeno; consideriamo una partizione di S, ossia una successione* $(B_n)_{n\in N}$ *di insiemi a 2 a 2 disgiunti e tali che la loro unione dia proprio S; fissiamo un certo evento A; la probabilità che A si verifichi può essere calcolata mediante la formula*

$$P(A)=\sum_{i=1}^{N} P(A\cap B_i)$$

Vediamo come questo teorema ci è di aiuto nel nostro caso. Intanto, lo spazio degli eventi relativo al fenomeno della generazione, da parte della sorgente, del segnale binario è semplicemente $S=\{0T,1T\}$, in quanto la sorgente può trasmettere o 0 o 1. Allora, come partizione di S noi possiamo prendere gli eventi

$$B_1=\{1T\}$$
$$B_2=\{0T\}$$

In tal modo, possiamo valutare la probabilità che si verifichi un errore mediante la formula

$$P(E)=P(E\cap 0T)+P(E\cap 1T)$$

Usando adesso le probabilità condizionate, possiamo esprimere le probabilità assolute in termini appunto di probabilità condizionate, ottenendo che

$$P(E)=P(E|0T)P(0T)+P(E|1T)P(1T)$$

Adesso riflettiamo su quanto abbiamo scritto:

- $P(E|0T)$ è la probabilità che avvenga un errore nell'ipotesi di aver trasmesso uno 0; in altre parole, è la probabilità che sia ricevuto 1 avendo trasmetto 0, ossia è la prima probabilità di transizione $p=p_0=P(1R|0T)$;

- in modo analogo, $P(E|1T)$ è la probabilità che avvenga un errore nell'ipotesi di aver trasmesso uno 1, ossia la probabilità che sia ricevuto 0 avendo trasmetto 1, ossia è la seconda probabilità di transizione $p=p_1=P(0R|RT)$;

Quindi abbiamo che

$$P(E)=pP(0T)+pP(1T)=p[P(0T)+P(1T)]$$

Inoltre, è chiaro che il termine tra parentesi quadre è la probabilità dell'evento $P(0T\vee 1T)$: infatti, l'evento 0T è disgiunto dall'evento 1T (in quanto la sorgente o tramette 1 o trasmette 0), per cui la somma delle probabilità di due eventi disgiunti è pari alla probabilità dell'evento unione. Tuttavia, l'evento $0T\vee 1T$ è l'evento certo, in quanto la sorgente o trasmette 1 o trasmette 0, per cui $P(0T\vee 1T)=1$.

IMPORTANZA DELLA RILEVAZIONE DEGLI ERRORI: BIT ERROR RATE

Generalmente, la probabilità che il canale commetta un errore è abbastanza piccola. Per caratterizzare questo aspetto delle prestazioni del canale, si definisce la cosiddetta **Bit Error Rate** (brevemente **BER**), ossia la *frequenza di errore sui bit*: essa rappresenta ogni quanti bit il canale commette un errore. I valori generalmente assunti dal BER sono compresi tra 10^{-7} (1 bit sbagliato su 10 milioni) e 10^{-3} (1 bit sbagliato su 1000). Apparentemente, si tratta di valori molto bassi, ma essi vanno confrontati con la velocità di trasmissione: per esempio, dato un canale avente BER=10^{-7}, se la velocità di trasmissione è di 10^9 bit/secondo, il numero medio di bit sbagliati in 1 secondo risulta essere

$$N = (\text{velocità}) \cdot \text{BER} = 10^9 \cdot 10^{-7} = 100$$

Avremmo dunque 100 errori al secondo (su 10^9 bit trasmessi in totale) e si tratta di una quantità tutt'altro che trascurabile. Si deduce l'importanza di ridurre al minimo la probabilità di errore: è cioè necessario trovare un metodo che riesca a rilevare ed eventualmente anche "recuperare" gli errori del canale.

CODICI A RIPETIZIONE (CENNI)

I metodi possibili per la rilevazione ed eventualmente la correzione degli errori di trasmissione sono diversi. Il primo che consideriamo è il cosiddetto "**codice a ripetizione**". La caratteristica fondamentale di questo metodo è quella per cui, dato il segnale emesso dalla sorgente, esso, prima di essere inviato al canale per la trasmissione, viene arricchito di un numero prefissato di bit, posti ovviamente in posizione opportuna. Vediamo di spiegarci meglio.

Intanto, dobbiamo modificare lo schema del sistema di tramissione nel modo seguente:

Sono stati dunque aggiunti due nuovi apparati, denominati "codificatore di canale" e "decoficatore di canale". Vediamo quali sono le loro funzioni a partire dal primo.

Supponiamo che la sorgente emetta la seguente sequenza di simboli binari:

0 1 0 0 1 1 1 0 0 0

Come si vede dallo schema, questa sequenza, prima di arrivare al canale per la trasmissione, va in input al "**codificatore di canale**": il compito di questo organo è quello di modificare il segnale binario nel modo seguente: per ogni simbolo binario in ingresso, il codificatore ne genera un numero DISPARI di copie. Nel nostro caso, in corrispondenza di quel segnale, il segnale generato dal canale, nell'ipotesi di 3 copie ogni bit, sarà il seguente:

000 111 000 000 111 111 111 000 000 000

Questo è il segnale che va nel canale e da questo deve essere trasmesso verso il ricevitore e questo è anche il motivo per cui si parla di "**codice a ripetizione**".

Nell'ipotesi che il canale non commetta nessun errore durante la trasmissione, quello è anche il segnale che arriva in ingresso al "**decodificatore di canale**" : il compito di questo organo è quello di ricostruire, a partire da questo segnale in ingresso, il segnale originale emesso dalla sorgente e di inviarlo quindi al ricevitore.

Come avviene la ricostruzione? Si dice che il decodificatore "***agisce a maggioranza***": per ogni terna di simboli che arriva in ingresso, il decodificatore trasmette al ricevitore il simbolo che, tra i 3, si ripete più volte. Nel caso di 3 ripetizioni, è ovvio che trasmetterà il simbolo che si ripete almeno 2 volte; nel caso generale di 2n+1 ripetizioni, il decodificatore trasmette il simbolo che si ripete almeno n+1 volte.

Vediamo allora per quale motivo questo metodo di trasmissione consente di ridurre la probabilità di errore, ossia la probabilità che il ricevitore riceva un simbolo diverso da quello effettivamente inviato dalla sorgente.

Sappiamo che il problema deriva dal canale, il quale, in modo del tutto casuale, può modificare qualcuno dei bit che riceve in ingresso dal codificatore. Per esempio, supponiamo che, anzichè trasmettere il segnale prima indicato, il canale trasmetta quest'altro:

$$0\underset{\uparrow}{1}0\ \ 1\underset{\uparrow}{0}\underset{\uparrow}{0}\ \ 000\ \ 000\ \ 11\underset{\uparrow}{0}\ \ 111\ \ \underset{\uparrow}{0}11\ \ 000\ \ 000\ \ 000$$

Le frecce indicano i bit in corrispondenza dei quali si è verificato un errore rispetto al segnale corretto. Vediamo come il decodificatore ricostruisce il segnale da inviare al ricevitore:

$$\underbrace{010}_{0}\ \underbrace{100}_{0}\ \underbrace{000}_{0}\ \underbrace{000}_{0}\ \underbrace{110}_{1}\ \underbrace{111}_{1}\ \underbrace{011}_{1}\ \underbrace{000}_{0}\ \underbrace{000}_{0}\ \underbrace{000}_{0}$$

Confrontando questo con il segnale emesso dalla sorgente, osserviamo che si è verificato 1 solo errore (nella seconda terna), a fronte di ben 5 errori commessi da parte del canale. Quindi, dei 5 errori, ben 4 sono stati recuperati.

E' dunque intuitivo accorgersi che la probabilità che il ricevitore riceva un simbolo sbagliato diventa senz'altro piccola: infatti, è chiaro che questa eventualità si verifica se il canale commette almeno 2 errori in 1 sola terna, il che è abbastanza difficile. Non solo, ma è anche intuitivo aspettarsi che la probabilità di errore si riduca di parecchio aumentando il numero di ripetizioni, ossia associando, a ciascun bit emesso dalla sorgente, un numero (dispari) ancora maggiore di copie.

Queste considerazioni intuitive possono essere ovviamente supportate da apposite formule. Per arrivare a queste formule, facciamo due ipotesi di partenza:

- la prima è che il canale sia simmetrico: abbiamo visto prima che questo significa che la probabilità di sbagliare simbolo è identica sia quando viene trasmetto uno 0 sia quando viene trasmesso un 1 ed abbiamo anche visto quali conseguenze ci siano per la valutazione della probabilità di errore;
- la seconda ipotesi è che un eventuale errore, da parte del canale, su di un bit non sia influenzata da altri errori e non influenzi altri errori; detto in altre parole, gli eventuali errori sono tutti eventi indipendenti tra loro.

Consideriamo allora l'evento E_S (detto **errore di sistema**) per cui il ricevitore riceve un bit diverso da quello inviato dalla sorgente: la probabilità che si verifichi l'errore di sistema (che, ricordiamo, è diversa da p, che è invece la probabilità di errore del canale sul generico bit) risulta avere espressione

$$P(E_S) = \sum_{k=n+1}^{2n+1} \binom{2n+1}{k} p^k (1-p)^{2n+1-k}$$

Trattandosi di una sommatoria finita di termini, una volta scelto n, ossia scelto il numero di ripetizioni (che è 2n+1), è chiaro che possiamo valutare numerivamente $P(E_S)$. Sostituendo i valori numerici, supponendo $p=10^{-2}$, si ottiene quanto segue:

n	$P(E_S)$
1	$3*10^{-4}$
2	10^{-5}
3	$3.5*10^{-7}$
4	$1.28*10^{-8}$
…	…

Questa tabella evidenzia come la probabilità di errore decresca notevolmente all'aumentare di n: per esempio essa ci dice che, per 5 ripetizioni (ossia n=4) la probabilità di errore vale $1.28*10^{-8}$, ossia, in altre parole, il ricevitore riceve un simbolo sbagliato approssimativamente ogni 10^8 simboli.

Naturalmente, ci sono anche degli svantaggi nel codice a ripetizione: infatti, è chiaro che quanti più simboli devono essere inviati, tanto maggiore è il tempo necessario alla trasmissione, per cui l'abbassamento della probabilità di errore, mediante il metodo del codice a ripetizione, comporta un innalzamento del tempo di trasmissione.

PROBABILITÀ DI ERRORE

Il codice a ripetizione analizzato nel paragrafo precedente è un tipico esempio in cui i bit non vengono trasmessi uno alla volta, ma a *blocchi* (o *pacchetti* o **parole**). Diventa allora importante studiare la probabilità che si verifichino uno o più errori sulla singola parola di bit trasmessa.

Indichiamo dunque con p la probabilità che 1 bit sia corretto e con q=1-p la probabilità che 1 bit sia sbagliato. Supponiamo di trasmettere i bit a blocchi di 2 e consideriamo il generico blocco:

- l'evento corrispondente ad avere entrambi i bit corretti equivale a due eventi che devono verificarsi contemporaneamente: il primo bit deve essere corretto (probabilità p) ed anche il secondo deve essere corretto (probabilità p): quindi, la probabilità che entrambi i bit siano corretti è $p \cdot p = p^2$;
- al contrario, per avere un bit sbagliato su 2 ci sono due possibilità: la prima è che il primo bit sia corretto (probabilità p) ed il secondo sbagliato (probabilità q) e la seconda è che il primo bit sia sbagliato (probabilità q) ed il secondo corretto (probabilità p): di conseguenza, la probabilità che ci sia un solo bit sbagliato su 2 è $pq + qp = 2pq$;
- infine, l'evento corrispondente ad avere entrambi i bit sbagliati si verifica quando il primo bit è sbagliato (probabilità q) ed anche il secondo bit è sbagliato (probabilità q), per cui la probabilità che entrambi i bit siano sbagliati è $q \cdot q = q^2$

Considerando che p+q=1, si osserva allora una cosa interessante: se calcoliamo il quadrato di (p+q), otteniamo

$$1 = (p+q)^2 = p^2 + 2pq + q^2$$

I termini dello sviluppo del quadrato di (p+q) corrispondono esattamente alle tre probabilità prima elencate, nello stesso ordine. Il risultato è in realtà generale: trasmettendo parole di codice binario composte da N bit, i coefficienti dello sviluppo del termine $(p+q)^N$, presi nell'ordine indicato dal triangolo di Tartaglia, forniscono le probabilità di avere, rispettivamente, 1,2,..., N errori su ciascuna parola.

Consideriamo ad esempio parole da N=4 bit (ossia le parole necessarie a codificare le 10 cifre decimali): abbiamo che

$$1=(p+q)^4=p^4+4p^3q+6p^2q^2+4pq^3+q^4$$

I termini di questo sviluppo hanno i seguenti significati:

p^4: probabilità di avere 4 bit corretti
$4p^3q$: probabilità di avere 3 bit corretti ed 1 sbagliato
$6p^2q^2$: probabilità di avere 2 bit corretti e 2 sbagliati
$4pq^3$: probabilità di avere 1 bit corretto e 3 sbagliati
q^4: probabilità di avere 4 bit sbagliati

Accertato questo, diventa importante valutare anche numericamente le probabilità appena elencate, al fine di capire quale situazione si verifichi più spesso (cioè quale situazione sia più probabile). Supponiamo, allora, che il canale che stiamo analizzando abbia probabilità di errore sul singolo bit q=0.4 (cui corrisponde ovviamente p=0.6): con questi valori, si trova che

p^4= 0.1296
$4p^3q$= 0.3456
$6p^2q^2$= 0.3456
$4pq^3$= 0.1536
q^4= 0.0256

Da questi valori si deduce che la situazione meno probabile è quella di avere 4 errori su 4, mentre quella più probabile è quella di avere o 1 o 2 errori. Si tratta, perciò, di un caso abbastanza critico, nel quale la probabilità di non avere errori (p^4) e superiore solo alla probabilità di avere 4 errori (q^4).

Questo risultato deriva in effetti dall'aver considerato un valore di q particolarmente alto, mai verificato nella realtà. Una situazione molto più ragionevole è invece quella di un canale la cui probabilità di errore sul singolo bit valga $q=10^{-6}$ (1 bit sbagliato su ogni milione). Con questo valore si ottengono i seguenti risultati:

$p^4=0.999996\cong p$
$4p^3q\cong 4*10^{-6}$
$6p^2q^2\cong 6*10^{-12}$
$4pq^3\cong 4*10^{-18}$
$q^4\cong 10^{-24}$

In questo caso, l'evento più probabile è quello di ottenere tutti i bit corretti, mentre la probabilità di avere errori scende molto rapidamente all'aumentare del numero di errori stessi. In particolare, le probabilità di avere 3 o 4 errori sono decisamente trascurabili e c'è inoltre una grossa differenza tra la probabilità di errore singolo ($4*10^{-6}$) e quella di errore doppio ($6*10^{-12}$). Consideriamo, ad esempio, un canale con velocità di trasmissione 10^9(bit/sec): con questa velocità e con le probabilità prima calcolate, l'errore singolo si verifica mediamente $10^9\cdot 4\cdot 10^{-6}=4000$ volte al secondo, mentre l'errore doppio si verifica mediamente $10^9\cdot 6\cdot 10^{-12}=0.006$ volte al secondo, ossia 6 volte ogni millisecondo.

Questi valori indicano che in una situazione del genere è di fondamentale importanza rilevare (ed eventualmente correggere) l'errore singolo, mentre si possono anche trascurare gli altri errori, che si verificano con una frequenza estremamente più piccole (6 ordini di grandezza).

Questo spiega il motivo per cui i prossimi paragrafi sono dedicati allo studio dei codici di rilevazione (e correzione) dell'errore singolo.

CODICE A CONTROLLO DI PARITÀ

Un concetto generale, già emerso nel paragrafo sul codice a ripetizione, è quello per cui ***i codici a correzione (e/o rilevazione) di errore prevedono l'uso di un numero di bit maggiore rispetto a quello strettamente necessario per la codifica dell'alfabeto sorgente.*** Consideriamo, per esempio, i codici numerici, cioè i codici binari necessari a codificare le 10 cifre decimali: abbiamo visto che, per codificare tali 10 cifre decimali, sono sufficienti parole di codice a 4 bit. Allora, per effettuare la rilevazione (ed eventualmente la correzione) dell'errore saranno utilizzate parole di almeno 5 bit. Vedremo che il numero di bit in più (cioè la cosiddetta **ridondanza**) determina il tipo di azione che si può realizzare in fase di ricezione.

I codici maggiormente utilizzati per la rilevazione dell'errore singolo sono i "**codici a controllo di parità**": sintenticamente, si tratta di aggiundere 1 bit a ciascuna parola del codice in modo tale che il numero di 1 presenti sia pari (**parità pari**) oppure dispari (**parità dispari**).

Facciamo ad esempio riferimento al codice BCD:

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

A ciascuna delle 10 parole di questo codice dobbiamo aggiungere un bit in modo tale il numero di bit 1 sia o pari o dispari: se il numero di bit a 1 è già dispari, il bit di parità sarà 0, mentre, in caso contrario, il bit di parità sarà 1. Per esempio, se vogliamo che il numero di bit a 1 sia dispari, avremo quanto segue:

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

La posizione del bit di parità può essere sia in testa (bit in posizione 4) sia in coda (bit in posizione 0). L'importante è che il ricevitore conosca a priori tale posizione.

Ovviamente, introducendo il bit di parità le parole trasmesse diventano ciascuna di 5 bit, per cui le probabilità di errore vanno ricalcolate ponendo N=5:

$$1 = (p+q)^5 = p^5 + 5p^4q + 10p^3q^2 + 10p^2q^3 + 5pq^4 + q^5$$

Prendendo nuovamente $q = 10^{-6}$, si ottengono i seguenti valori:

$p^5 = 0.999995 \cong p$
$5p^4q \cong 5q = 5*10^{-6}$
$10p^3q^2 \cong 10q^2 = 10^{-11}$
$10p^2q^3 \cong 10q^3 = 10^{-17}$
$5pq^4 \cong 5*10^{-24}$
$q^5 \cong 10^{-30}$

E' ovvio che, una volta formata e trasmessa la parola da 5 bit (incluso quindi il bit di parità), l'errore si può verificare anche sul bit di parità.

Quando la generica parola di codice arriva al ricevitore, esso controlla la parità e si possono allorap presentare le seguenti situazioni:

- se si è verificato un errore singolo, la parità risulta violata, per cui il ricevitore si accorge della presenza di un errore; esso non può però decidere quale dei 5 bit pervenuti sia sbagliato, per cui non è in grado di effettuare la correzione: può semplicemente segnalare la presenza dell'errore ed eventualmente chiedere la ritrasmissione della parola;
- se si è verificato un errore doppio, la parità non risulta violata, per cui il ricevitore non può accorgersi della presenza dei due errori: questo è l'inconveniente fondamentale di questo codice. Si deduce, in particolare, che esso può sempre rilevare la presenza di errori su un numero dispari di bit (1,3 o 5 errori), mentre non può rilevare la presenza di errori su un numero pari di bit (2 o 4 errori).

Considerando, però, che l'errore quadruplo è altamente improbabile, mentre l'errore doppio è abbastanza meno probabile dell'errore singolo, la conclusione è che il codice a controllo di parità conferisce una apprezzabile immunità dall'errore singolo.

Il secondo inconveniente di questo codice è naturalmente quello di richiedere maggiori tempi di elaborazione e maggiori tempi di trasmissione, data evidentemente la presenza di un ulteriore bit in ciascuna parola di codice. Sono inoltre richiesti circuiti più complessi (visto che ci sono da trattare più bit simultaneamente) e memorie più capienti.

C'è anche uno svantaggio legato alla ridondanza: infatti, mentre con 5 bit sono teoricamente rappresentabili 2^5=32 possibili configurazioni, il codice ne usa solo 10. Vengono cioè usate solo 1/3 delle possibili configurazione, ossia c'è una ridondanza di 3 ad 1.

D'altra parte, questo vale nel caso in cui il codice utilizzato preveda parole da 4 bit (cui va aggiunto il bit di parità). Si può invece verificare che, ***all'aumentare della lunghezza delle parole di codice, la ridondanza diminuisce:*** per esempio, consideriamo il **codice ASCII**, che prevedere parole da 8 bit, cui corrispondono 2^8=256 combinazioni, tutte utilizzate. Aggiungendo il bit di parità otteniamo parole da 9 bit, ossia 2^9=512 combinazioni: di queste, sono sempre 256 quelle utilizzate (cioè la metà), per cui la ridonanza è di 2 ad 1. Rispetto ad un codice a 4 bit, la ridondanza è evidentemente diminuita.

Ridurre la ridondanza significa in definitiva migliorare l'efficienza del sistema complessivo, ossia minimizzare le conseguenze (in termini di complessità circuitale, tempo di elaborazione, tempo di trasmissione e così via) dovute all'introduzione del bit di parità.

Ecco perchè esistono dei codici che utilizzando intrinsecamente il controllo di parità: si trata cioè di codici le cui parole possiedono già di per sé, senza alcuna aggiunta, il controllo di parità. Uno di questi è il cosiddetto **codice 2 out of 5**: si tratta di un codice, con parole di 5 bit, usato ancora una volta per la codifica delle 10 cifre decimali. L'espressione "*2 out of 5*" indica il fatto per cui, su tutte le 32 possibili configurazioni con 5 bit, quelle che presentano solo 2 bit posti ad 1 sono esattamente 10: infatti $\frac{5!}{2!\cdot 3!}=10$.

Avendo quindi a disposizione 10 parole con qesta caratteristica, esse vengono scelte per rappresentare le 10 cifre decimali.

Esistono degli appositi circuiti che ricevono in ingresso le parole di codice BCD e forniscono in uscita le corrispondenti parole del codice 2 OUT OF 5.

Osservazione

(richiami di Teoria dei Segnali) Il codice a controllo di parità può anche essere usato per elaborare una generica sequenza di bit, senza preoccuparsi del suo significa. Supponiamo ad esempio che la sequenza di bit da trasmettere sia la seguente:

0 0 1 0 1 1 1 0 0 1 0 1 0 1 1

La prima operazione compiuta dal codificatore di canale consiste nel raggruppare i bit della sequenza da trasmettere in gruppi di 2n-1 bit, dove ovviamente il valore di n dipende dal progetto. Per esempio, prendiamo n=2, per cui i gruppi che otteniamo sono composti da 3 bit e, nel nostro esempio, sono i seguenti:

$\underbrace{0\ 0\ 1}\ \underbrace{0\ 1\ 1}\ \underbrace{1\ 0\ 0}\ \underbrace{1\ 0\ 1}\ \underbrace{0\ 1\ 1}$

A ciascuno di questi gruppi viene aggiunto un ulteriore **bit di parità**, ad esempio a destra: questo bit ha lo scopo di rendere PARI o DISPARI il numero di bit a 1 contenuti in ciascun gruppo. Per esempio, volendo sempre un numero pari di bit posti ad 1, il bit aggiuntivo verrà posto a 1 quando il numero di bit a 1 è dispari, mentre, in caso contrario, verrà posto a 0.

Nel nostro esempio, la sequenza viene allora modificata nel modo seguente:

$\underbrace{0\ 0\ 1\ 1}\ \underbrace{0\ 1\ 1\ 0}\ \underbrace{1\ 0\ 0\ 1}\ \underbrace{1\ 0\ 1\ 0}\ \underbrace{0\ 1\ 1\ 0}$

Questa è dunque la sequenza che il codificatore di canale invia sul canale stesso per la trasmissione.

Passiamo alla fase di ricezione. In particolare, supponiamo che si verifichino degli errori durante la trasmissione e che quindi la sequenza che giunge al decodificatore di canale sia fatta nel modo seguente:

$\underbrace{0\ 0\ 1\ 1}\ \underbrace{0\ \overset{\downarrow}{0}\ 1\ 0}\ \underbrace{1\ \overset{\downarrow}{1}\ \overset{\downarrow}{1}\ 1}\ \underbrace{0\ 0\ 1\ 0}\ \underbrace{0\ 1\ 1\ 0}$

Le freccette nel disegno indicano evidentemente gli errori che si sono avuti durante la trasmissione.

Il decodificatore controlla che, per ciascun gruppo di n bit (dove 2n-1 bit sono i bit di informazione, mentre l'n° bit è quello di parità), il numero di bit ad 1 sia pari:

- quando il numero di bit ad 1 è pari, il decodificatore prende per buono il gruppo, elimina il bit di parità e trasmette il gruppo al ricevitore;
- quando il numero di bit ad 1 è dispari, il decodificatore deduce che c'è stato un errore.

Da quanto detto si deduce che si possono verificare, in ricezione, tre diverse situazioni:

- *ricezione corretta* : quando, sul generico gruppo di n bit, non si è verificato alcun errore, è chiaro che il gruppo è corretto, per cui esso viene inviato al ricevitore (privato del bit di parità);
- *errore non rilevato* : quando, sul generico gruppo di n bit, si è verificato un numero PARI di errori, il gruppo contiene un numero pari di bit ad 1, per cui il decodificatore accetta il gruppo e lo trasmette al ricevitore; la differenza con il caso precedente è che la sequenza è in questo caso sbagliata, ossia è diversa da quella emessa dalla sorgente;
- *errore rilevato* : quando, sul generico gruppo di n bit, si è verificato un numero DISPARI di errori, il gruppo non contiene certamente un numero pari di bit ad 1, per cui il decodificatore deduce la presenza di almeno un errore e quindi chiede la "*ritrasmissione*" del gruppo.

Analizziamo allora da un punto di vista probabilistico queste eventualità, facendo sempre l'ipotesi di avere a disposizione un **CBS** (canale binario simmetrico) con probabilità di errore pari a p:

- la "ricezione corretta" si ha quando, su n bit, non si verifica nessun errore, per cui, applicando la *formula di Bernoulli*, possiamo scrivere che

$$P\begin{pmatrix}\text{ricezione}\\ \text{corretta}\end{pmatrix} = (1-p)^{2n}$$

- l' "errore non rilevato" si ha invece quando il numero di errori, su n bit, è pari: abbiamo dunque che

$$P\begin{pmatrix}\text{errore non}\\ \text{rilevato}\end{pmatrix} = \sum_{k=0}^{n}\binom{2n}{2k}p^{2k}(1-p)^{2n-2k}$$

- in modo analogo, l' "errore rilevato" si verifica quando il numero di errori, su n bit, è dispari, per cui

$$P\begin{pmatrix}\text{errore}\\ \text{rilevato}\end{pmatrix} = \sum_{k=0}^{n}\binom{2n}{2k-1}p^{2k-1}(1-p)^{2n-2k+1}$$

CORREZIONE DELL’ERRORE SINGOLO

Abbiamo detto che per ideare un codice che consenta la rilevazione dell’errore singolo è necessario aumentare la ridondanza, ossia fondamentalmente la lunghezza di ciascuna parola del codice binario. Se oltre alla rilevazione si vuole effettuare anche la correzione, la ridondanza va ulteriormente aumentata. Per esempio, come vedremo tra poco, se si vuole poter correggere l’errore singolo in un codice numerico, servono ben 7 bit (mentre abbiamo visto che, per la semplice rilevazione dell’errore singolo, ne erano sufficienti 4): con 7 bit, si hanno 128 possibili combinazioni, per cui, dato che ne vengono usate solo 10, la ridonanza è di circa 13 ad 1.

Al fine di costruire un codice che permetta la correzione dell’errore singolo, si usa il concetto di **distanza** tra le parole del codice considerato: date due parole di codice consecutive, la loro distanza è il numero di bit di cui differiscono. Consideriamo per esempio il codice BCD:

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

In generale, la minima distanza è ovviamente 1 in tutti i codici, in quanto una distanza 0 implicherebbe due parole uguali, il che non renderebbe il codice univocamente decodificabile. Nel caso del codice BCD, tale minima distanza vale proprio 1. La massima distanza è invece pari a 4 e corrisponde alla distanza tra le parole corrispondenti al 7 ed all’8. Ci sono poi anche distanze intermedie di 2 e di 3.

La minima distanza tra due parole di codice è detta **distanza di Hamming**: nel caso del codice BCD abbiamo appena detto che vale 1, ma ci sono codici in cui essa è maggiore di 1. Se introducessimo un bit per il controllo di parità, si verifica facilmente che la distanza di Hamming diventa 2:

	BCD	bit di parità	Parola finale
0	0000	1	00001
1	0001	0	00010
2	0010	0	00100
3	0011	1	00111
4	0100	0	01000
5	0101	1	01011
6	0110	1	01101
7	0111	0	01110
8	1000	0	10000
9	1001	1	10011

Come si nota dalla tabella, non ci sono parole consecutive che differiscono di 1 solo bit, ma, per esempio, le parole corrispondenti a 0 e ad 1 (oppure quelle corrispondenti ad 8 e 9) differiscono di 2 bit, per cui la distanza di Hamming vale effettivamente 2.

Il concetto di distanza di Hamming è in pratica il seguente: *se due parole di codice hanno una distanza di Hamming d, saranno necessari d errori sui bit per passare da una all’altra.*

Si dimostrano a questo punto due risultati fondamentali:

Teorema- *La minima distanza di Hamming necessaria per poter rilevare d errori su singoli bit è d+1, mentre la minima distanza di Hamming necessaria per poter correggere d errori su singoli bit è 2d+1*

Possiamo giustificare facilmente il primo di questi due risultati: dire che il codice considerato ha una distanza di Hamming pari a d+1, significa dire che ci vogliono almeno d+1 errori perchè da una generica parola si possa passare ad un'altra: di conseguenza, se si verificano solo d errori, la parola che si ottiene non apparterrà sicuramente al codice e quindi potrà essere individuata come parola sede di errori.

Giustificare il secondo risultato è invece più difficile. Cominciamo col dire che noi siamo interessati alla correzione dell'errore singolo: ponendo quindi d=1, il teorema ci dice che la minima distanza di Hamming necessaria per la correzione dell'errore singolo è 3.

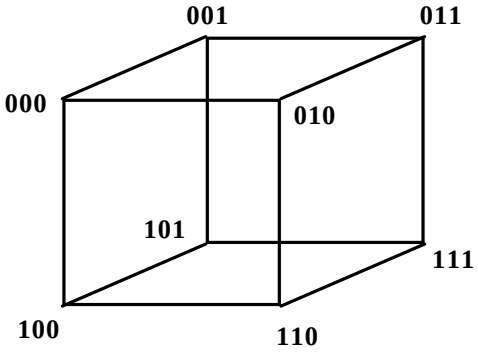
Facciamo adesso riferimento ad un caso concreto. Consideriamo un codice con parole da 3 bit, cioè composto da 8 possibili combinazioni:

000 001 010 011 100 101 110 111

Possiamo disporre queste 8 parole agli 8 vertici di un cubo. In particolare, le possiamo disporre in modo tale che la "distanza" tra due vertici qualsiasi sia sempre pari a 1: per fare questo, si usa il già citato metodo delle linee di riflessione, che porta al seguente risultato

000 001 011 010 110 111 101 100

Disponiamo queste parole ai vertici di un cubo:



Partendo da un qualsiasi vertice, le parole situate nei tre vertici adiacenti hanno sempre una distanza pari ad 1. Si ha inoltre quanto segue:

- partendo da un qualsiasi vertice e percorrendo 3 spigoli, si ottiene sempre una parola con distanza 3 dalla parola del vertice di partenza;
- partendo da un qualsiasi vertice e percorrendo 3 diagonali, si ottiene sempre una parola con distanza 2 dalla parola del vertice di partenza;

Sulla base di queste conclusioni, facciamo il seguente ragionamento: consideriamo una parola qualsiasi, ad esempio 100; se su questa parola si verifica un errore singolo, il ricevitore riceve sicuramente una delle 3 parole poste sui vertici adiacenti a quello corrispondente a 100, ossia riceve 101 oppure 110 oppure 000. In ogni caso, quindi, non viene comunque ricevuta una parola che sia a distanza 3 dalla parola originale 100.

Supponiamo allora che le uniche due parole di codice siano 100 e l'unica parola, tra le altre 7, che dista 3 da 100, ossia la parola 011: queste due parole vengono inserite in una tabella, detta **look-up table**, a disposizione del ricevitore. Vediamo allora se il ricevitore è in grado di effettuare la correzione. Facciamo l'ipotesi che sia stata trasmessa la parola 100 e che sia arrivata la parola 101, per cui si è verificato un errore sul terzo bit. Il ricevitore, una volta rilevato l'errore, si comporta nel modo seguente:

- ipotizza che sia il primo bit ad essere sbagliato e lo corregge, ottenendo 001: questa parola non fa parte delle possibili parole di codice (che sono solo 100 e 011), per cui deduce che il primo bit non è sbagliato;
- ipotizza allora che sia sbagliato il secondo bit, per cui lo corregge, ottenendo 111: nemmeno questa parola fa parte delle possibili parole di codice, per cui deduce che neanche il secondo bit può essere sbagliato;
- conclude allora che può essere sbagliato solo il terzo bit, per cui lo corregge ed ottiene 100, cioè la parola corretta.

Abbiamo cioè trovato che la correzione funziona solo a patto che le parole di codice considerate abbiano distanza 3, a conferma di quanto enunciato dal teorema precedentemente esposto.

CODICE HAMMING

Gli eventuali bit di parità usati per rilevare e correggere l'errore singolo vengono messi in posizione diversa da codice a codice, a seconda dell'azione che si intende effettuare. Un caso in cui la posizione dei bit di parità corrisponde ad uno scopo ben preciso è quello del **codice di Hamming sistematico**: come vedremo tra un attimo, *la posizione dei bit viene scelta in modo tale da individuare, proprio tramite il loro valore, la posizione del bit sbagliato.* Vediamo di spiegarci meglio.

Il punto di partenza è dunque un codice binario composto da parole di n bit: nel caso dei codici numerici, n vale 4. Partendo da queste parole di codice, vogliamo formare un altro codice che consenta di rilevare e correggere l'errore singolo: per ottenere questo scopo, aggiungiamo un numero k (per il momento generico) di bit per il controllo di parità, per cui le parole di codice diventano di lunghezza n+k. Il primo obbiettivo che ci poniamo è quello di determinare, fissato n, il minimo valore di k necessario.

Il discorso è semplice: i k bit di parità devono servire, tramite k distinte **prove di parità**, a individuare con esattezza la posizione del bit sbagliato: dato che l'errore singolo si può presentare sia tra gli n bit che contengono l'informazione sia sui k bit di parità, è chiaro che i possibili errori singoli sono n+k, per cui i k bit devono poter indicare n+k posizioni diverse. D'altra parte, può anche presentarsi il caso in cui non c'è nessun errore ed i k bit di parità devono poter indicare anche questa situazione: concludiamo perciò che ci sono n+k+1 situazioni possibili e i k bit di parità devono poter indicare tutte queste n+k+1 situazioni. Allora, dato che con k bit è possibile indicare 2k situazioni diverse, il vincolo da imporre è

$$2^k \geq n + k + 1$$

Questa relazione non è risolvibile analiticamente, ma solo per tentativi. Usiamo una tabella per indicare i vari tentativi:

n	k	n+k+1	2^k	accettabilità
1	1	3	2	NO
1	2	4	4	SI
2	2	6	4	NO
2	3	6	8	SI
3	3	7	8	SI
4	3	8	8	SI
5	4	10	16	SI
...	4	...	16	SI
11	4	16	16	SI

Il primo caso è n=1. Provando con k=1 bit di controllo, la relazione $2^k \geq n + k + 1$ non è verificata, per cui dobbiamo provare con k=2: in questo caso, il vincolo è soddisfatto, per cui deduciamo immediatamente che per un codice con parole di lunghezza n=1, bastano 2 bit di parità per effettuare la correzione dell'errore singolo.

Passiamo ad n=2: evidentemente, ponendo k=2 la relazione $2^k \geq n + k + 1$ non è verificata, mentre lo è per k=3: deduciamo che per un codice con parole di lunghezza n=2, bastano 3 bit di parità per effettuare la correzione dell'errore singolo.

Sempre 3 bit di parità bastano anche per parole di codice di lunghezza 3 ed anche di lunghezza 4 (quindi per i codici numerici), mentre invece serve un 4° bit di parità per parole di codice aventi lunghezza compresa tra 5 ed 11 e così via.

Si osserva, allora, un risultato molto importante: man mano che la lunghezza n delle parole di codice aumenta, il rapporto k/n si riduce, ossia aumenta l'**efficienza** del codice a correzione di errore, intesa appunto come reciproco del rapporto tra il numero di bit di codice ed il numero di bit di parità:

- per n=1, risulta k/n=2, ossia una efficienza di 1/2;
- per n=2, risulta k/n=1.5, cioè una efficienza di 2/3;
- per n=3, risulta k/n=1, cioè una efficienza unitaria;
- ...
- per n=11, risulta k/n=4/11, ossia una efficienza di 11/4.

In definitiva, quindi, *un codice a controllo di parità è tanto più efficiente, dal punto di vista del rapporto k/n, quanto maggiore è il numero n di bit che contengono l'informazione vera e propria.*

In generale, un codice binario le cui parole contengono n bit di dati e k bit di controllo si indica con la sigla **nB(n+k)B**, in modo appunto da indicare la lunghezza iniziale del codice e quella con l'aggiunta dei bit di parità.

Premesso tutto questo, dobbiamo adesso capire come posizionare i k bit di controllo nella generica parola di codice, come effettuare le prove di parità, come rappresentare l'esito di tali prove mediante i k bit ed infine come interpretare il significato di questi k bit.

Per semplicità, facciamo come al solito riferimento ad un codice numerico, avente perciò parole di lunghezza n=4. In base alla tabella di prima, il numero k di bit per il controllo di parità, se si vuole la correzione dell'errore singolo, è 3, per cui le parole di codice assumono lunghezza 3:

7	6	5	4	3	2	1

Le posizioni dei singoli bit si contano in verso crescente, andando da destra verso sinistra e partendo da 1: il bit più a destra è dunque quello in posizione 1, il bit centrale è in posizione 4 ed il bit più a sinistra è quello in posizione 7.

I 3 bit di parità, che indichiamo con p_1,p_2,p_3, vanno sistemati nelle posizioni corrispondenti a potenze di 2: la posizione 1 corrisponde a 2^0=1 ed ospiterà il bit 1, la posizione 2 corrisponde a 2^1=2 e ospiterà il bit p_2 e la posizione 4, che corrisponde a 2^2=4, ospiterà il bit p_3:

7	6	5	p_3 (4)	3	p_2 (2)	p_1 (1)

Gli altri 4 bit sono invece quelli destinati all'informazione: se $b_4b_3b_2b_1$ è la parola di codice prima dell'introduzione dei bit di controllo, la sistemazione dei 4 bit è quella indicata nello schema seguente:

b_4 (7)	b_3 (6)	b_2 (5)	p_3 (4)	b_1 (3)	p_2 (2)	p_1 (1)

Il motivo per cui viene adottata questa disposizione dei bit è il seguente: una volta effettuate, nel modo che vedremo, le 3 prove di parità e una volta attribuiti, in base agli esiti di tale prove, i valori ai 3 bit di parità, *la parola $p_3p_2p_1$ composta dai tre bit di parità indicherà, tramite il suo valore decimale, la posizione del bit eventualmente errato:* ad esempio, se la parola $p_3p_2p_1$ risulta essere 101 (5 in decimale), questo significa che il bit errato è quello in posizione 5, ossia b_2. Nella tabella seguente sono indicate tutte le possibilità:

p_3	p_2	p_1	decimale	errore in ..
0	0	0	0	nessun bit
0	0	1	1	posizione 1
0	1	0	2	posizione 2
0	1	1	3	posizione 3
1	0	0	4	posizione 4
1	0	1	5	posizione 5
1	1	0	6	posizione 6
1	1	1	7	posizione 7

Affinchè si possa ottenere questa indicazione dai bit di parità, è necessario effettuare le prove di parità in modo molto particolare:

- cominciamo dalla **1° prova di parità**: bisogna verificare che la parola composta dai bit in posizione *7,5,3,1* (quindi $b_4b_2b_1p_1$) contenga un numero dispari (o pari a seconda della convenzione scelta a priori) di 1; se questo accade, allora il bit p_1 va posto a 0, mentre, in caso contrario, esso va posto ad 1;
- passiamo alla **2° prova di parità**: bisogna verificare che la parola composta dai bit in posizione *7,6,3,2* (quindi $b_4b_3b_1p_2$) contenga un numero dispari (o pari) di 1; se questo accade, allora il bit p_2 va posto a 0, mentre, in caso contrario, esso va posto ad 1;
- infine, passiamo alla **3° prova di parità**: bisogna verificare che la parola composta dai bit in posizione *7,6,5,4* (quindi $b_4b_3b_2p_3$) contenga un numero dispari (o pari) di 1; se questo accade, allora il bit p_3 va posto a 0, mentre, in caso contrario, esso va posto ad 1;

Come si osserva, alcuni bit vengono verificati in più prove di parità (come il bit in posizione 7, che compare in tutte e 3 le prove), mentre altri solo in una sola prova di parità (come i bit in posizione 1,2 e 4, ossia i 3 bit di parità).

Facciamo allora degli esempi di come si applicano i procedimenti appena descritti.

Consideriamo la parola 1001 del codice BCD (rappresentativa del 9 decimale) e cominciamo a disporre i suoi bit sulla parola da 7 bit che dovremo generare (in sede di codifica) e successivamente trasmettere:

7	6	5	4	3	2	1
1	**0**	**0**	$\mathbf{p_3}$	**1**	$\mathbf{p_2}$	$\mathbf{p_1}$

Dobbiamo ora attribuire i valori dei tre bit di parità:

- il bit p_1 serve al controllo di parità dei bit in posizione 7531: i valori dei bit in corrispondenza di tali posizioni sono $101p_1$; affinchè questa parola contenga un numero dispari di 1, è evidente che dobbiamo porre p_1=1:

7	6	5	4	3	2	1
1	**0**	**0**	$\mathbf{p_3}$	**1**	$\mathbf{p_2}$	**1**

- il bit p_2 serve invece al controllo di parità dei bit in posizione 7632: i valori dei bit in corrispondenza di tali posizioni sono $101p_2$; affinchè questa parola contenga un numero dispari di 1, è evidente che dobbiamo porre p_2=1:

7	6	5	4	3	2	1
1	**0**	**0**	$\mathbf{p_3}$	**1**	**1**	**1**

- il bit p_3 serve infine al controllo di parità dei bit in posizione 7654: i valori dei bit in corrispondenza di tali posizioni sono $100p_3$; questa parola contiene già un numero dispari di 1, per cui dobbiamo questa volta porre p_3=0:

7	6	5	4	3	2	1
1	**0**	**0**	**0**	**1**	**1**	**1**

La parola da trasmettere è dunque 1000111. Se questa parola arriva così com'è al ricevitore (o meglio al decodificatore posto prima di esso), è ovvio che le tre prove di parità daranno tutte esito negativo, cioè 000, da cui il ricevitore dedurrà che non si è verificato alcun errore.

Supponiamo invece che si sia verificato un errore ad esempio in posizione 5, per cui la parola che giunge al ricevitore è 1010111. Il ricevitore va dunque ad eseguire le tre prove di parità:

- **1° prova di parità** la parola composta dai bit in posizione *7,5,3,1* risulta essere 1111 e contiene un numero pari di 1: di conseguenza, il ricevitore attribuisce 1 all'esito di tale prova;
- **2° prova di parità** la parola composta dai bit in posizione *7,6,3,2* risulta essere 1011 e contiene un numero dispari di 1, per cui il ricevitore attribuisce 0 all'esito di tale prova;
- **3° prova di parità** la parola composta dai bit in posizione *7,6,5,4* risulta essere 1010 e contiene un numero pari (o pari) di 1, per cui il ricevitore attribuisce 1 all'esito di tale prova.

La parola che risulta dunque formata dagli esiti delle tre prove di verità è 101 (cioè 5 decimale), da cui il ricevitore deduce che si è verificato un errore ed è sul bit in posizione 5, che quindi può essere corretto.

Facciamo un altro esempio, supponendo di voler trasmettere la parola 1100 (non appartenente al codice BCD):

7	6	5	4	3	2	1
1	**1**	**0**	$\mathbf{p_3}$	**0**	$\mathbf{p_2}$	$\mathbf{p_1}$

Andiamo ad attribuire i valori dei tre bit di parità:

- il bit p_1 serve al controllo di parità dei bit in posizione 7531: i valori dei bit in corrispondenza di tali posizioni sono $100p_1$, per cui $p_1=0$:

$$\underset{7}{\mathbf{1}} \mid \underset{6}{\mathbf{1}} \mid \underset{5}{\mathbf{0}} \mid \underset{4}{\mathbf{p_3}} \mid \underset{3}{\mathbf{0}} \mid \underset{2}{\mathbf{p_2}} \mid \underset{1}{\mathbf{0}}$$

- il bit p_2 serve invece al controllo di parità dei bit in posizione 7632: i valori dei bit in corrispondenza di tali posizioni sono $110p_2$, per cui $p_2=1$:

$$\underset{7}{\mathbf{1}} \mid \underset{6}{\mathbf{1}} \mid \underset{5}{\mathbf{0}} \mid \underset{4}{\mathbf{p_3}} \mid \underset{3}{\mathbf{0}} \mid \underset{2}{\mathbf{1}} \mid \underset{1}{\mathbf{0}}$$

- il bit p_3 serve infine al controllo di parità dei bit in posizione 7654: i valori dei bit in corrispondenza di tali posizioni sono $110p_3$, per cui $p_3=1$:

$$\underset{7}{\mathbf{1}} \mid \underset{6}{\mathbf{1}} \mid \underset{5}{\mathbf{0}} \mid \underset{4}{\mathbf{1}} \mid \underset{3}{\mathbf{0}} \mid \underset{2}{\mathbf{1}} \mid \underset{1}{\mathbf{0}}$$

La parola da trasmettere è dunque 1101010. Supponiamo che si verifichi un errore sul bit in posizione 2 (cioè sul bit di parità p2), per cui la parola che giunge al ricevitore è 1101000. Il ricevitore va dunque ad eseguire le tre prove di parità:

- **1° prova di parità**: la parola composta dai bit in posizione $7,5,3,1$ risulta essere 1000, per cui l’esito della prova è 0;
- **2° prova di parità**: la parola composta dai bit in posizione $7,6,3,2$ risulta essere 1100, per cui l’esito della prova è 1;
- **3° prova di parità**: la parola composta dai bit in posizione $7,6,5,4$ risulta essere 1101, per cui l’esito della prova è 0.

La parola che risulta dunque formata dagli esiti delle tre prove di verità è 010 (cioè 2 decimale), da cui il ricevitore deduce che si è verificato un errore sul bit in posizione 2.

Abbiamo dunque capito come si impostano i bit di parità e come si effettuano i test di parità in fase di ricezione. Ci chiediamo adesso cosa succede se sulla parola trasmessa si sono verificati due errori. Per capirlo, facciamo un esempio concreto: supponiamo che la parola da trasmettere, con i bit di parità già impostati, sia quella dell’esempio precedente, ossia

$$\underset{7}{\mathbf{1}} \mid \underset{6}{\mathbf{1}} \mid \underset{5}{\mathbf{0}} \mid \underset{4}{\mathbf{1}} \mid \underset{3}{\mathbf{0}} \mid \underset{2}{\mathbf{1}} \mid \underset{1}{\mathbf{0}}$$

Supponiamo che ci siano due errori, in posizione 6 ed in posizione 2, per cui il ricevitore riceve

$$\underset{7}{\mathbf{1}} \mid \underset{6}{\mathbf{0}} \mid \underset{5}{\mathbf{0}} \mid \underset{4}{\mathbf{1}} \mid \underset{3}{\mathbf{0}} \mid \underset{2}{\mathbf{0}} \mid \underset{1}{\mathbf{0}}$$

Il ricevitore esegue le tre prove di parità:

- **1° prova di parità** (7,5,3,1) → 1000 → esito 0;
- **2° prova di parità** (7,6,3,2) → 1000 → esito 0;
- **3° prova di parità** (7,6,5,4) → 1001 → esito 1.

La parola ottenuta è dunque 100 (4 in decimale), per cui il ricevitore si convince che il bit sbagliato è quello in posizione 4 e lo corregge, ottenendo 1000000. Questa parola, rispetto a quella effettivamente corretta (cioè quella giusta), possiede 3 errori: è successo, quindi, che l'errore è stato rilevato, ma la correzione ha peggiorato le cose.

Questo, quindi, per dire che *il controllo di parità rileva sia l'errore singolo sia l'errore doppio, ma può correggere solo il primo.* Se si volesse ottenere anche la correzione dell'errore doppio, sarebbe necessario complicare il codice tramite un aumento della ridondanza. D'altra parte, abbiamo visto che l'errore singolo è decisamente più probabile dell'errore doppio, per cui ideare un codice per la correzione anche dell'errore doppio è una scelta da farsi solo in casi critici in cui anche l'errore doppio non è tollerabile.

Autore: SANDRO PETRIZZELLI
e-mail: sandry@iol.it
Sito personale: http://users.iol.it/sandry
Succursale: http://digilander.iol.it/sandry1