APPUNTI DI INFORMATICA

di Andrea de Prisco

Moduli, comunicazioni e protocolli

Questo mese parleremo di comunicazioni, intese come scambio di messaggi e dati tra le varie unità di un sistema di calcolo. Distingueremo prima di tutto fra trasmissioni seriali e parallele. Vedremo poi due tipi classici di scambio messaggi (sincrono e asincrono), infine vedremo tre esempi di protocolli di trasmissione studiati per segnalare possibili errori verificatisi durante il trasferimento, eventualmente correggendoli, nonché un codice a lunghezza variabile che permetterà di inviare un insieme di simboli da un modulo a un altro in minor tempo.

Comunicazione tra moduli

La prima cosa che faremo prima di iniziare a parlare di comunicazioni, sarà quella di porci a un livello di astrazione superiore a quello che generalmente siamo abituati a osservare. Nella prima puntata di Appunti di informatica abbiamo più volte sottolineato che un calcolatore va sempre visto come un insieme più o meno grande di parti tra loro interagenti: abbiamo visto che in un calcolatore c'è sempre una memoria, una o più unità di processo per eseguire i programmi degli utenti e «necessariamente» alcune unità di ingresso/uscita per l'interfacciamento col mondo esterno. In altre parole col mondo di noi uomini (e donne) che immettiamo nel calcolatore i programmi tramite una tastiera e leggiamo i risultati delle elaborazioni su carta o direttamente da un video.

Tutte queste unità, dicevamo, per poter interagire hanno bisogno di una vera e propria forma di dialogo: sempre nel primo «Appunt-amento» abbiamo visto come CPU e memoria se la intendono in quanto a operazioni di lettura o scrittura di informazioni in determinate celle.

Questo mese ci eleveremo a considerare tutte queste unità interagenti, semplicemente come dei moduli capaci di comunicare con altri moduli attraverso dei canali. Come illustrato in figura 1, avremo ad esempio il modulo A che dialoga col modulo B, e non staremo a specificare cosa essi effettivamente rappresentano dato che quanto vedremo vale per tutti.

La freccia tra i due moduli rappresenta il canale di comunicazione e il suo orientamento indica in quale verso avviene il dialogo. Generalmente canali bidirezionali tra due moduli (fig. 2A) sono implementati semplicemente utilizzando due canali distinti (fig. 2B), uno dal modulo X al modulo Y, l'altro dal modulo Y al modulo X.

Tornando alla figura 1, se A intende dialogare con B eseguirà un'operazione di invio messaggio al modulo B. Analogamente B, per prelevare il messaggio che A intende inviargli esegue un'operazione di ricevi messaggio. Cosa succede se nell'attimo in cui A manda il suo messaggio a B, questo non sia disposto a riceverlo essendo magari impegnato a fare qualcos'altro? E che succede se B vuole ricevere il messaggio da A e questo non glielo manda?

Figura 1

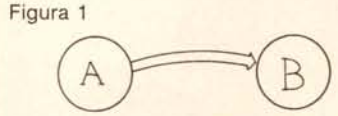


Figura 2A

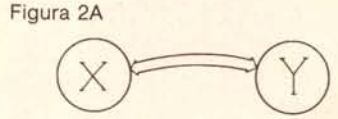


Figura 2B

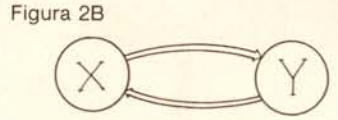
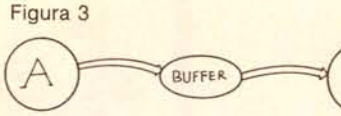


Figura 3



Distinguiamo due tipi di comunicazione.

Sincrona e Asincrona

Immaginiamo che il modulo A sia un modulo produttore (cioè in qualche modo produce informazione da inviare ad altri, ad esempio un'unità a dischi che legge dati dal supporto magnetico) mentre il modulo B sia un consumatore (ossia riceve messaggi da altri, ad esempio una stampante ove il messaggio in questione sia una linea da stampare). La sequenza delle operazioni eseguite dai due moduli sarà allora del tipo che A prepara un messaggio, lo manda a B, B lo preleva (farà qualcosa con questo); A prepara un nuovo messaggio, B lo preleva e così via fino ad esaurimento.

Se la comunicazione è sincrona, l'attimo in cui A manderà un messaggio a B e questo lo preleverà sarà lo stesso. In altre parole A lo manda e non fa nulla finché B non l'ha prelevato. Analogamente se B chiede un messaggio prima che A l'abbia mandato, semplicemente aspetta. È come se due amici si dessero appuntamento per consegnarsi qualcosa: chi prima arriva aspetta, ma l'attimo in cui il qualcosa passa da un amico all'altro è unico. Per questo motivo tale tipo di comunicazione viene detta anche a rendezvous stretto.

La comunicazione asincrona, come è facile immaginare, permette di non far aspettare nei limiti del possibile né

destinatario né mittente. È il caso del postino che quotidianamente lascia la posta in cassetta, sia che è presente il destinatario in casa, sia che è assente. Per realizzare tale meccanismo tra moduli comunicanti, si introduce un modulo aggiuntivo buffer (fig. 3) che funge da parcheggio per i messaggi in arrivo, restituendoli nello stesso ordine ad ogni richiesta da parte del modulo consumatore. Il suo funzionamento è abbastanza semplice: il modulo buffer ha a disposizione una memoria ed è collegato in ingresso col modulo A e in uscita col modulo B. Se A gli manda un messaggio, effettuerà con questo un rendez-vous stretto e prelevato il messaggio lo metterà nella sua memoria. Analogamente con B effettuerà altri rendez-vous stretti tutte le volte che questo richiederà messaggi. È ovvio che se la frequenza dei messaggi inviati è molto più alta di quella di prelevamento il buffer può riempirsi completamente e dovremo far aspettare ugualmente A. Analogamente se B chiede più messaggi di quanti A gliene manda nell'unità di tempo, il Buffer si svuota completamente e ad aspettare sarà B. Da ciò si evince che il tutto funziona bene se il buffer ha una capacità abbastanza grande da non riempirsi facilmente e B non va troppo di corsa rispetto ad A. In ogni caso anche con un buffer ad una sola posizione, cioè in grado di mantenere un solo messaggio, si riesce sempre a slegare le operazioni di invio e prelevamento tra due moduli.

Seriale o parallela?

Finora abbiamo visto un po' quello che accade dal punto di vista logico nella comunicazione tra due o più moduli comunicanti. Prima di passare alle vere e proprie strategie di trasmissione dati occorre fare una ben precisa distinzione tra trasmissione seriale e parallela delle singole informazioni. Questo perché alcune strategie sono proprie di un modello mentre altre sono bivalenti.

Le informazioni, come visto nella prima puntata, all'interno di un calcolatore sono mantenute sottoforma di numeri binari (ossia composti di soli 0 e 1) in quanto per i circuiti digitali è più facile riconoscere queste due sole cifre, associando per esempio allo 0 la mancanza di differenza di potenziale rispetto alla massa e alla cifra 1 generalmente una d.d.p. pari a circa 5 volt.

Si tratta a questo punto di definire una corrispondenza biunivoca tra informazioni e insieme di cifre binarie, in modo da poter far circolare, di fatto, non solo 0 e 1 dentro a un calcolatore ma anche simboli dell'alfabeto così come cose più complicate (array, record, liste ecc.).

Potremmo ad esempio associare alla lettera A il codice 0001, alla B 00010 alla C il codice 00011 e così via, per trattare col computer testi letterari.

Anche per i trasferimenti da un modulo a un altro, si tratta di far viaggiare una serie di 1 e di 0. Col trasferimento seriale manderemo uno dopo l'altro su un'unica linea i bit (cifre binarie) che compongono i caratteri da trasferire. Col trasferimento parallelo, se usiamo codici a n bit, useremo n linee, una per ogni bit, in modo da inviare tutto d'un fiato un intero carattere alla volta.

Vantaggi e svantaggi sono ovvi: nel primo caso il collegamento sarà più semplice in quanto è richiesto un solo filo su cui far viaggiare il messaggio; nel secondo caso, potremo effettuare trasferimenti molto più alla svelta a scapito però dell'economicità del collegamento che come detto richiede n linee.

Integrità delle trasmissioni

A questo punto non ci resta che far dialogare tra loro due moduli, cercando di far arrivare al destinatario effettivamente quanto il mittente gli ha mandato: in due parole: evitare errori.

Come al solito si tratterà di scegliere un compromesso tra sicurezza e costo di una trasmissione, che sembra proprio essere proporzionale al numero di bit inviati. Infatti se oltre al messaggio vero e proprio mandiamo alcuni bit di controllo, possiamo dare la possibilità al mittente di scoprire un eventuale errore verificatosi durante la trasmissione o addirittura di correggerlo. La cosa che può accadere normalmente è che qualche bit, inviato come 0, lungo il percorso cambi, presentandosi al destinatario come un 1, o viceversa. Anche se questo accade abbastanza raramente, specie se il percorso da compiere non è lungo (ricordiamo ai lettori che esistono reti di calcolatori dette geografiche le quali hanno nodi-computer anche in continenti diversi) non bisogna sottovalutare questa eventualità.

Il primo metodo per la segnalazione d'errore è stato quello del «bit di parità». Consiste nell'aggiungere un bit ad ogni singola informazione inviata, posto a 0 o a 1 a seconda dell'informazione stessa. Per non cadere in strani equivoci, ricordiamo che una informazione è una qualsiasi entità esprimibile da un calcolatore reale, es.: un numero intero, un carattere.

Distinguiamo fra bit di parità pari e bit di parità dispari. Nel primo caso tale bit è posto a 1 se il numero totale di bit 1 del dato sono dispari, 0 se il numero totale di bit a 1 del dato è pari. Nel caso della parità dispari funziona al contrario, ma è formalmente la stessa cosa. In pratica, inviando un dato col protocollo di bit di parità pari saremo certi di inviare una quantità pari di 1 (fra quelli del dato e il bit di parità). Se il destinatario riceverà una quantità dispari di 1 vorrà dire che qualcosa non ha funzionato sulla linea e un bit si è invertito. Per completezza diamo un'occhiata alla figura 4. In A il byte inviato contiene un numero dispari di 1 (per l'esattezza 5), poniamo il bit di parità a 1 ottenendo in tutto 6 bit a 1 quindi una quantità pari. In B abbiamo il caso di parità dispari quindi per ottenere in tutto una quantità dispari di 1 lasceremo il bit di parità a 0.

Certo potrebbe verificarsi che due bit cambino stato durante un viaggio e in tale situazione il bit di parità non segnalerebbe l'errore, perché se erano pari gli uno restano pari e se erano dispari restano dispari. Ma ciò è davvero molto raro che avvenga e poi, per definizione, un codice a segnalazione d'errore ci informa se qualcosa non va, non se tutto è andato bene: in altre parole se il numero di bit a 1 passa da pari a dispari o viceversa c'è stato sicuramente un errore di trasmissione,

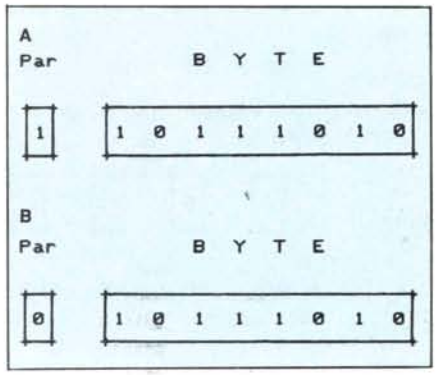


Figura 4 - In (A) poniamo il bit di parità ad 1 portando così il numero totale di 1 ad una quantità pari. In (B) ponendo il bit di parità a 0, abbiamo una quantità dispari di bit a 1.

se invece ciò non accade non possiamo avere la certezza che tutto sia a posto.

Detto questo, prima di far vedere che esistono codici a segnalazione e correzione d'errore, capovolgiamo la situazione e mostriamo come sia possibile, a scapito questa volta della sicurezza, risparmiate sul numero totale di bit necessari per trasferire, ad esempio, un testo letterario dalla memoria di un calcolatore ad una stampante.

Il codice Huffman

Questo codice di trasmissione è detto anche «dipendente dalla frequenza» in quanto per poter fruire dei suoi benefici (buona questa!) è necessario che i caratteri da trasferire non abbiano frequenza simile: deve accadere ad

```
A B C D D A A E A D
C C A D A D A B G B
D A A B A B E D A A
A A B A A A A E A D
B B A B A C B A B D
A A E A C A A C F D
E B B B A B C A B G
A B A B A C A F G B
A C B A B A F A F H
B B A C B B A F A H
```

Figura 5A - Immaginiamo di voler trasferire questo insieme di caratteri.

esempio che alcuni caratteri siano più frequenti e altri meno frequenti. Fortunatamente ciò accade anche nella lingua italiana, dove consonanti come la Q sono certamente più rare della S o della T o roba simile.

Oltre a questo, per poter funzionare, il codice Huffman necessita di trasferimento seriale che, come detto prima, invia un bit dopo l'altro. A questo punto passiamo alla figura 5A e come dice la didascalia immaginiamo di voler trasferire tale insieme di simboli. Da una veloce scorsa si nota subito una forte maggioranza di caratteri come la A e la B, contro un numero limitato di G e di H. Siamo in un caso abbastanza ottimale per applicare il codice Huffman. Senza di questo, come detto prima, dovremmo associare biunivocamente una sequenza di 0 e 1 ad ogni simbolo, per esempio così:

```
A   000
B   001
C   010
D   011
E   100
F   101
G   110
H   111
```

E trasferendo l'insieme dei 100 simboli di 5A invieremmo in tutto 300 bit (100 simboli ×3 bit l'uno). Vediamo di fare qualcosa di meglio: seguite sempre in figura 5. Il primo passo consiste nell'elencare tutti i simboli del testo ponendo sotto di essi la relativa occorenza (fig. 5B).

Abbiamo infatti 40 A, 25 B, 10 C ecc.

Si tratta di costruire l'albero di figura 5F procedendo nel seguente modo: (fig. 5C) si fondono le occorrenze più piccole, 3 e 2 rispettivamente dei caratteri G e H ottenendo così una nuova occorrenza (5) dei caratteri G e H considerati insieme. Si itera il procedimento scegliendo sempre le occorrenze più piccole, considerando anche le occorrenze ottenute per fusione. Ad esempio, dopo le fusioni di fig. 5E le occorrenze da scegliere saranno 5 e 10 ottenendo così l'occorrenza 15.

Alla fine otterremo l'occorrenza 100 che è naturalmente quella dei simboli A...H, che vuol dire appunto che i simboli dalla A alla H sono in tutto presenti 100 volte, ovvio.

Ultimo passo, etichettare con un 1 tutti i rami sinistri dell'albero e con uno 0 tutti i rami destri come indicato in fig. 5F. Possiamo ora generare il codice Huffman di quei caratteri: partendo dalla occorrenza massima, raggiungiamo i vari caratteri prendendo nota del percorso che facciamo ossia prelevando gli 1 o gli 0 che incontriamo. Quindi alla lettera A associeremo il codice 1, al B il codice 01 (infatti per risalire da 100 fino alla B, occhio alla figura, dobbiamo percorrere prima un ramo destro poi un ramo sinistro). Tutto il codice sarà dunque:

```
A    1          Tot.   1 bit
B    01          "     2   "
C    0011        "     4   "
D    0010        "     4   "
E    00011       "     5   "
F    00010       "     5   "
G    00001       "     5   "
H    00000       "     5   "
```

Se proviamo a trasferire il testo ci accorgeremo di aver risparmiato: infatti noi dobbiamo trasferire:

```
40 A = 40 x 1 =  40 bit +
25 B = 25 x 2 =  50 bit +
10 C = 10 x 4 =  40 bit +
10 D = 10 x 4 =  40 bit +
 5 C =  5 x 5 =  25 bit +
 5 C =  5 x 5 =  25 bit +
 3 C =  3 x 5 =  15 bit +
 2 C =  2 x 5 =  10 bit =
-------------------------
        totale = 245 bit
```

Come dire che ne abbiamo risparmiato ben 55 pari a più del 18%.

Per quel che riguarda la decodifica, ossia il ritornare ai caratteri man mano che si ricevono i bit serialmente, notiamo che anche questo è molto semplice, infatti come per i numeri te-

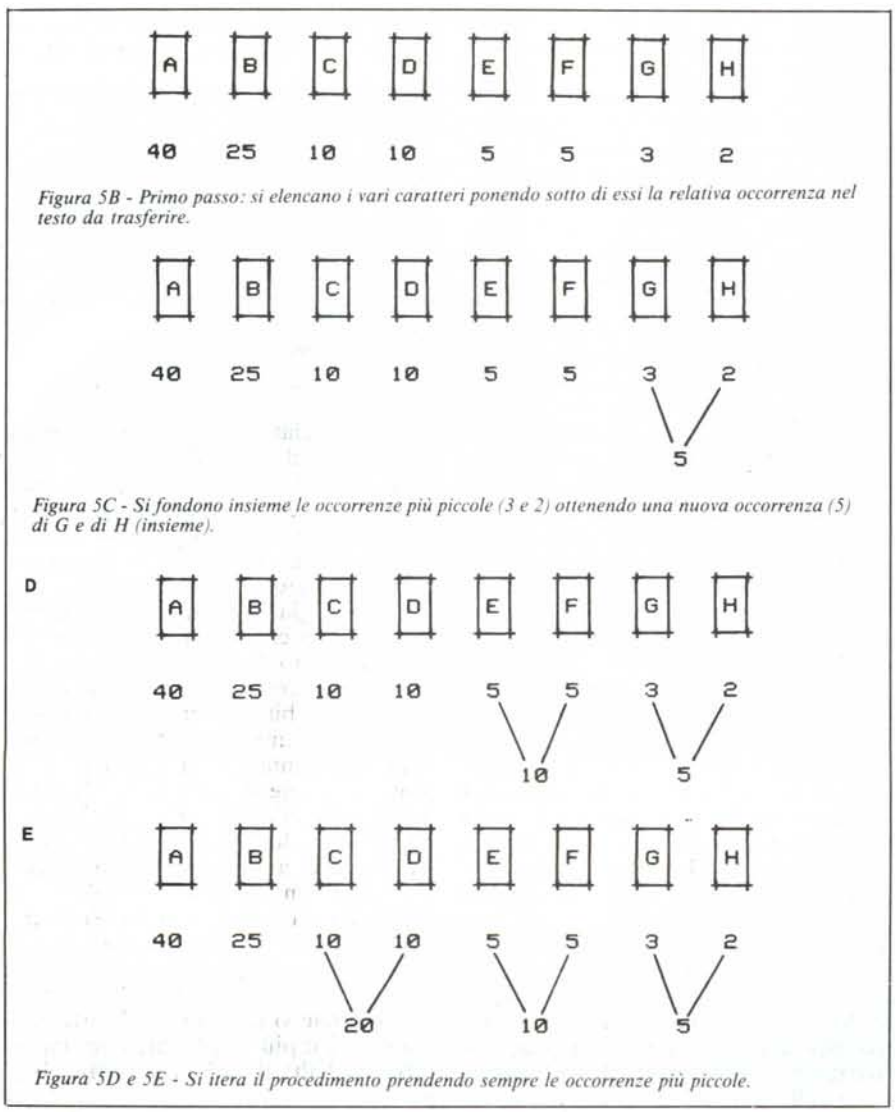


Figura 5B - Primo passo: si elencano i vari caratteri ponendo sotto di essi la relativa occorrenza nel testo da trasferire.

Figura 5C - Si fondono insieme le occorrenze più piccole (3 e 2) ottenendo una nuova occorrenza (5) di G e di H (insieme).

Figura 5D e 5E - Si itera il procedimento prendendo sempre le occorrenze più piccole.

lefonici non capita mai che un abbonato abbia un numero la cui parte iniziale sia a sua volta il numero di un altro abbonato, così nel nostro codice non vi sono rappresentazioni con tale tipo di ambiguità. Tornando all'esempio telefonico, non possono esistere due abbonati dello stesso distretto uno con numero 3139 e l'altro con numero 313933 in quanto altrimenti da un qualsiasi telefono una volta composte le prime 4 cifre, la centrale telefonica non sa se aspettare altre cifre o passare l'abbonato 3139. Nel nostro caso se riceviamo un 1 sappiamo che questo rappresenta una A; se riceviamo uno 0, non possiamo dire ancora nulla, vediamo cosa viene dopo: se riceviamo un 1 allora era una B, se riceviamo un altro 0 dobbiamo aspettare i prossimi 2 o 3 bit per decodificare: l'importante che la strada che seguiremo sull'albero è unica e quindi ci porta a un ben preciso carattere. Attenzione al fatto che se perdiamo anche un solo bit per la strada possiamo chiudere baracca e burattini perché cominceremo a leggere caratteri mai inviati così come codici inesistenti o roba simile, quindi, da questo punto di vista il metodo è assai labile.

Codici correttori d'errore

Qualcuno dei lettori ancora non ci crederà ma esistono effettivamente dei metodi per correggere automaticamente un errore di trasmissione: posto cioè che qualche bit da 0 passi a 1 o viceversa, il sistema è in grado di riconoscerlo e, naturalmente, di correggerlo complementandolo.

Dovremo chiaramente usare un po' di bit in più ma non tanti quanto si potrebbe pensare. Il metodo è dovuto a Richard Hamming che lo inventò nell'ormai lontanissimo 1950 (nell'informatica eravamo davvero agli albori a quei tempi).

Per trasferire byte di 8 bit dovremo impiegarne 12, non importa se poi il trasferimento avverrà in maniera seriale o parallela. Numereremo i nostri 12 bit, come mostrato in fig. 6 da sinistra verso destra (o da destra verso sinistra, non ha importanza) usando per lo scopo i numeri da 1 a 12 (non si può con numerazioni diverse, tipo 0...11).

A questo punto occorre notare che ciascuno dei numeri da 1 a 12 può essere generato da somme di potenze di 2 comprese per l'appunto tra 1 e 12. Tali potenze sono 1,2,4 e 8 e come è facile verificare sommando opportunamente questi valori si può ottenere qualsiasi bit. Ad esempio, 12 sarà dato da 8+4, 7 da 4+2+1 e così via.

I bit 1,2,4 e 8 saranno rispettivamente i bit di parità dei bit che essi stessi generano. Quindi il bit 1 (che genera 1,3,5,7,9,11) controllerà i bit 1,3,5,7,9,11 analogamente per i bit 2,4 e 8 come indicato in figura 6. Controllerà nel senso visto nel paragrafo «Integrità delle trasmissioni», con parità pari.

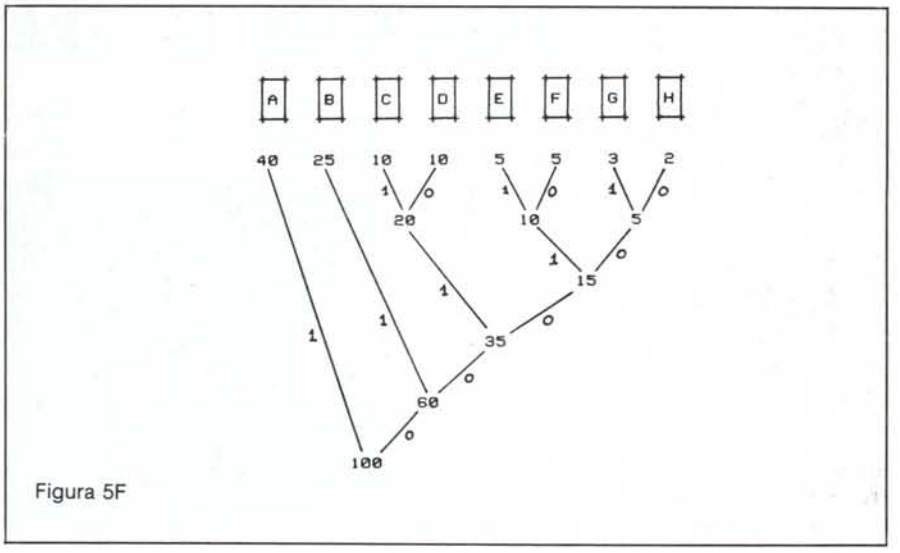


Figura 5F

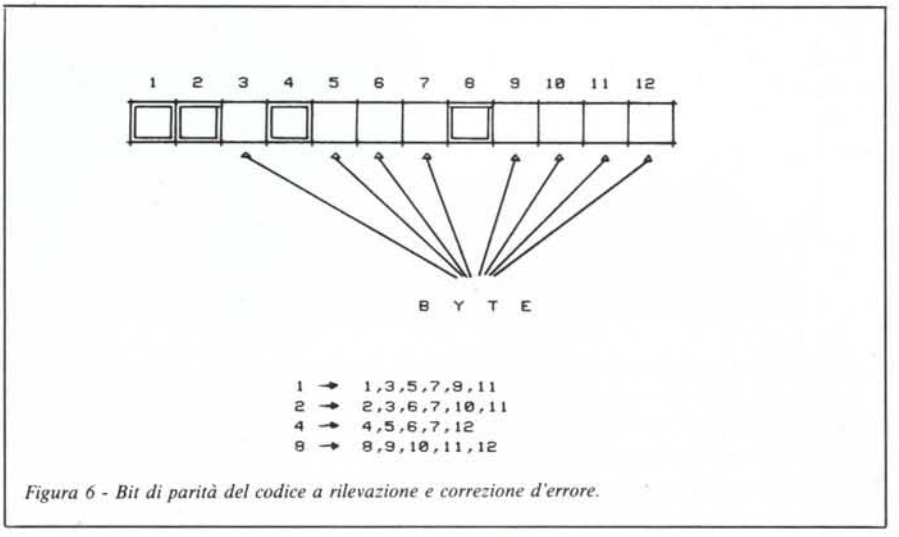


Figura 6 - Bit di parità del codice a rilevazione e correzione d'errore.

Questa costruzione intreccia in modo tale il controllo della parità che il bit eventualmente trasferito complementato segnalerà l'errore nei bit di parità che lo generano. Dimostrare questo non è una cosa molto breve, quindi cerchiamo di chiarire le eventuali perplessità con qualche esempio.

Sistemiamo il nostro byte nelle posizioni indicate in fig. 6 e settiamo i bit di parità tenendo presente l'indicazione sottostante, quindi si vede nei bit 3,5,7,9,11 quanti 1 ci sono e si sistema di conseguenza il bit 1; poi si contano gli 1 dei bit 3,6,7,10,11 e si setta il bit di parità 2. Analogamente per i bit 4 e 8. Il pacchetto così ottenuto possiamo mandarlo al destinatario il quale controllerà se qualche bit di parità non corrisponde a quanto ci si aspetta. Se accade questo, il bit sbagliato è quello dato dalla somma dei bit di parità sbagliati. Facciamo un esempio: supponiamo che il bit 3 (dei nostri 12) durante il viaggio transiti da 0 a 1 o viceversa. Il bit 3 è controllato, insieme ad altri, dal bit di parità 1 e 2 (vedi sempre fig. 6). Ed è proprio da questi che noi sentiremo puzza di bruciato. Come per magia notiamo che 1+2 è proprio 3, il bit che avevamo supposto essere sbagliato. Supponiamo invece che il bit 7 sia errato: essendo il bit 7 controllato dai bit di parità 1,2 e 4 questi ci segnaleranno l'errore e noi andremo a complementare il bit 1+2+4=7. Se dovesse essere sbagliato un solo bit di parità, ironia della sorte, sarà proprio questo (e solo questo) ad aver creato confusione: anche per lui una medicina: complementarlo.

Inutile dirvi che se in un unico trasferimento si presentano più complementazioni il nostro sistema fallisce, ma come detto prima, già è raro, molto raro, che vi sia un solo errore, figuriamoci 2 o più di 2 (... ma non impossibile, ih!, ih!, ih!... Firmato: Murphy).