Appunti di Calcolatori Elettronici
Interpretazione degli indirizzi di memoria

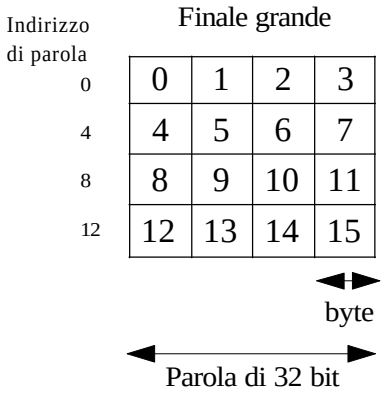
Introduzione: Big Endian e Little Endian

Ci chiediamo il modo in cui venga "interpretato" un **indirizzo di memoria**, ossia sostanzialmente quale sia l'*oggetto* che viene referenziato dall'indirizzo e dallo spiazzamento. Per rispondere a questa domanda, considereremo macchine la cui memoria gode delle seguenti due caratteristiche:

- è organizzata a byte;
- è possibile indirizzare i byte (8 bit), le parole corte (16 bit) e le parole (32 bit). In molti casi, sarà anche permesso l'indirizzamento delle parole doppie (64 bit).

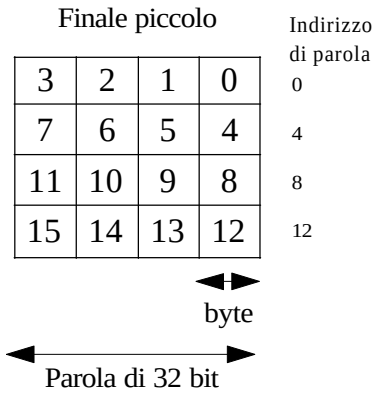
La considerazione iniziale da fare è la seguente: i byte che costituiscono una **parola** possono essere "numerati" da sinistra verso destra o viceversa. Questa potrebbe apparire come una scelta irrilevante, ma in realtà ci si accorge presto che non è così. Ad esempio, mentre nella famiglia **Motorola** (a 32 bit) i byte sono normalmente numerati da sinistra a destra, al contrario, nella famiglia **INTEL** (a 32 bit) avviene il contrario, ossia la numerazione è da destra verso sinistra. Nel primo caso di parla di *calcolatori a finale grande* (**Big Endian**) mentre nel secondo caso di parla di *calcolatori a finale piccolo* (**Little Endian**).

La figura seguente riporta 4 parole della memoria di un calcolatore a 32 bit, i cui byte sono numerati da sinistra a destra:



In questo caso, l'**indirizzo di parola** (0,4,8 o 12 nel nostro esempio) identifica il byte più a sinistra di ciascuna parola (byte più significativo).

La prossima figura riporta invece 4 parole della memoria di un calcolatore a 32 bit, i cui byte sono numerati da destra a sinistra:



In questo caso, l'indirizzo di parola (sempre pari a 0,4,8 o 12) identifica il byte più a destra di ciascuna parola (byte meno significativo). In definitiva, quindi, *nell'ordinamento di tipo Big Endian, l'indirizzo di un dato è quello del suo byte più significativo, mentre invece, nell'ordinamento di tipo Little Endian, l'indirizzo di un dato è quello del suo byte meno significativo*.

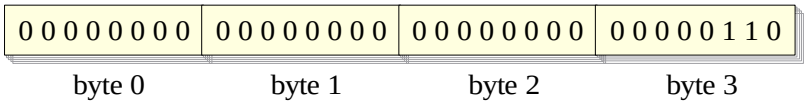
Naturalmente, però, i numeri vengono memorizzati sempre da destra verso sinistra: per esempio, supponiamo di avere il numero intero 6, il quale, espresso in binario con 32 bit, è

```
00000000 00000000 00000000 00000110
```

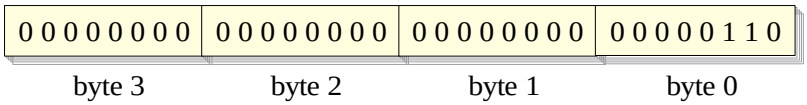
Nel caso del sistema *a finale grande*, questa configurazione viene inserita, così com'è, a partire dal byte 0 per finire al byte 3. Al contrario, nel caso del sistema *a finale piccolo*, il primo byte a sinistra va nel byte 3, il secondo a sinistra va nel byte

2, il successivo nel byte 1 e l'ultimo, quello che contiene i bit significativi, va nel byte 0:

Big Endian



Little Endian



In ogni caso, i bit vengono comunque riportati con i tre bit 110 a destra della parola, mentre i rimanenti 29 bit a sinistra sono tutti zeri.

Inoltre, in entrambi i casi, l'indirizzo della parola che contiene questo numero è sempre 0. In un certo senso, quindi, cambia l'ordine con cui i bit vengono "letti": *mentre nel finale grande si legge a partire dalla cella avente l'indirizzo specificato e si va per indirizzi crescenti, nel finale piccolo si fa il contrario*.

Trasferimento dati tra due PC con diverso ordinamento

Date le premesse del precedente paragrafo, vediamo perché l'ordinamento dei byte è un aspetto tutt'altro che trascurabile. Per semplicità, consideriamo una semplice "scheda personale" di un impiegato, costituita da una stringa (il nome dell'impiegato) e da due interi (l'età ed il numero del dipartimento di appartenenza):

```
Nome: Jim Smith
Età: 21 anni
Dipartimento: 260 (1×256+4=260)
```

Dovendo memorizzare questa scheda, abbiamo bisogno di 9 byte per il nome (ogni byte corrisponde ad un carattere, incluso lo spazio), di 1 byte per l'età e di 2 byte per il numero di dipartimento. Se poi vogliamo far riferimento a tali informazioni non a livello di singoli byte, ma a livello di singole parole (ad esempio da 32 bit), avremo bisogno di inserire degli 0 non significativi per riempire tali parole. Ad esempio, la rappresentazione a finale grande è mostrata nella figura seguente:

Finale grande

0	J	I	M	
4	S	M	I	T
8	H	0	0	0
12	0	0	0	21
16	0	0	1	4

Se invece vogliamo la rappresentazione a finale piccolo, abbiamo quanto segue:

Finale piccolo

				Indirizzo
	M	I	J	0
T	I	M	S	4
0	0	0	H	8
0	0	0	21	12
0	0	1	4	16

E’ chiaro che entrambe le rappresentazioni sono efficaci. Il problema nasce invece quando una delle due macchine cerca di mandare la scheda ad un’altra macchina appartenente alla stessa rete. Ad esempio, supponiamo che quella a finale grande spedisca la scheda a quella a finale piccolo, trasmettendo un byte alla volta, cominciando dal byte 0 e finendo con il byte 19 ([1]). In questa situazione, il byte 0 della numerazione a finale grande va nel byte 0 della memoria a finale piccolo, il byte 1 della numerazione a finale grande va nel byte 1 della memoria a finale piccolo e così via, dando origine alla seguente situazione nella memoria della macchina a finale piccolo:

Esito del trasferimento da finale grande a finale piccolo

				Indirizzo
	M	I	J	0
T	I	M	S	4
0	0	0	H	8
21	0	0	0	12
4	1	0	0	16

[1] Per semplicità, supponiamo che i bit dei byte non vengano invertiti durante la trasmissione.

Il problema sorto è evidente: quando la macchina a finale piccolo tenta di stampare la scheda, tutto funziona bene per il nome, mentre invece si ottiene una età di 21×2^{24} anni e l'indicazione del dipartimento è altrettanto erronea. Il motivo è che la trasmissione ha invertito l'ordine dei caratteri in tutte le parole, ma solo nelle prime tre questa operazione era lecita (trattandosi di stringhe), mentre invece non lo era nelle altre due (contenenti numeri interi).

Una soluzione ovvia del problema sarebbe quella di invertire, via software, i byte di una parola dopo aver fatto la copia:

Esito del trasferimento da finale grande a finale piccolo con scambio

				Indirizzo
J	I	M		0
S	M	I	T	4
H	0	0	0	8
0	0	0	21	12
0	0	1	4	16

Il problema, questa volta, è che i numeri interi sono giusti, ma lo sono le stringhe: infatti, quando il calcolatore legge la stringa iniziale, legge prima il byte 0 (che contiene uno spazio), poi il byte 1 (=M) e così via, commettendo ancora una volta un errore.

Si intuisce come una soluzione semplice al problema non ci sia. Una soluzione possibile, ma anche poco efficiente, potrebbe essere quella di includere un **header** (*testata*), all'inizio di ogni record, che specifichi che tipo di dati seguono (stringhe, interi, altri) e quanto sono lunghi; in tal modo, il calcolatore ricevente sa quando eseguire gli scambi e quando no.

Autore: **Sandro Petrizzelli**
e-mail: sandry@iol.it
sito personale: http://users.iol.it/sandry