La Rappresentazione dell'Informazione

Maurizio Palesi

Sommario

In questo documento sarà trattato il modo in cui, in un calcolatore, vengono rappresentati i vari generi di informazione (testi, numeri interi, numeri reali ecc.) mediante opportune tecniche di codifica.

Introduzione

Un problema rilevante che si deve affrontare nei processi di elaborazione dell'informazione è la *rappresentazione dell'informazione*. Questo documento saranno dapprima classificati i sistemi numerici in generale paricolarrizzando il caso dei sistemi numerici di tipo posizionale. Saranno presentate, utilizzando un approccio ad esempi, le tecniche di conversione da un sistema di numerazione ad un altro. Sarà presentata la rappresentazione sia dei numeri interi (naturali e relativi) sia dei numeri razionali. Inoltre sarà illustrato come in linguaggio C vengono rappresentati i tipi di dato semplici.

Sistemi Numerici

Un sistema numerico è determinato quando si fissano alcuni elementi che lo caratterizzano:

- Un insieme limitato di simboli (cifre) che rappresentano quantità intere prestabilite
- Le regola che devono essere applicate per costruire i numeri
  - Non Posizionali: il valore delle cifre è indipendente dalla loro posizione (es. M nella numerazione romana vale sempre 1000);
  - Posizionali: a ogni posizione all'interno della rappresentazione è associato un peso.

Di particolare importanza sono i sistemi numerici di tipo posizionale e per questo motivo saranno gli unici che considereremo.

Sistemi Numerici Posizionali

Nei sistemi numerici posizionali il numero (non negativo) *N* è rappresentato come una sequenza di *n* simboli d_i:

$$N = d_{n-1}d_{n-2}\ldots d_2d_1d_0$$

Dove ogni simbolo d_i viene chiamato *cifra*.

Nei sistemi numerici posizionali, ogni cifra ha un peso diverso a seconda della posizione che occupa. Per esempio nel sistema numerico decimale, ad ogni cifra viene assegnato un peso proporzionale ad una potenza del 10. Più precisamente, alla cifra i-esima viene assegnato un peso pari a 10^i. Per convenzione la cifra i-esima di un codice numerico è la i-esima cifra partendo da destra e iniziando a contare da 0. Per esempio se consideriamo il numero decimale 10061974 si ha:

1	**0**	**0**	**6**	**1**	**9**	**7**	**4**
7	6	5	4	3	2	1	0

La cifra di posizione 0 vale 4 e pesa 10^0. La cifra di posizione 1 vale 7 e pesa 10^1, la cifra di posizione 2 vale 9 e pesa 10^2. Generalizzando, la cifra di posizione *i* pesa 10^i. Il valore di un codice numerico si ottiene sommando i prodotti delle cifre per il rispettivo peso. Riprendendo l'esempio precedente si ha infatti:

$$10061974 = 1 \times 10^7 + 0 \times 10^6 + 0 \times 10^5 + 6 \times 10^4 + 1 \times 10^3 + 9 \times 10^2 + 7 \times 10^1 + 4 \times 10^0$$

Generalizzazione

In generale dato un qualsiasi codice posizionale in base *b* ogni cifra può assumere un simbolo appartenente ad un alfabeto ordinato di *b* simboli $\mathcal{D}$.

Dato il seguente codice posizionale in base *b*:

$$N_b = d_{n-1} d_{n-2} \ldots d_2 d_1 d_0$$

se ad ogni simbolo di $\mathcal{D}$ si assegna in modo univoco un numero intero compreso tra 0 e *b-1* allora l'equivalente di N_b in decimale (che indicheremo con N_{10}) viene così calcolato:

$$N_{10} = \sum_{i=0}^{n-1} d_i \times b^i \tag{1}$$

Esempi di Conversione in Decimale

Vedremo ora alcuni esempi di conversione da generiche basi *b* alla base 10 (decimale).

Da Binario a Decimale

In un codice binario, l'alfabero dei simboli utilizzato contiene solo due elementi: lo zero e l'uno (cioè $\mathcal{D} = \{0, 1\}$). Identificheremo tali simboli con il termine *bit*. Inoltre chiameremo bit meno significativo (LSB, Less Significative Bit) il bit di posizione 0 (o il primo bit da destra), e bit più significativo (MSB, most significative bit) il primo bit da sinistra.

Ricordendo la Formula 1 e sapendo che *b=2* il numero binario $N_2 = 0101101$ in decimale vale:

$$\begin{aligned} N_{10} &= 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = \\ &= 0 + 32 + 8 + 4 + 1 = 45 \end{aligned}$$

È possibile fare alcune osservazioni:

1. Un numero binario il cui LSB vale 0 è pari altrimenti è dispari. Infatti ricordando la Formula 1 si vede che il numero decimale corrispondente è una somma di potenze del 2 (e quindi un numero pari). L'ultimo addendo di questa somma è $d_0 \times 2^0$. Quindi se $d_0 = 0$ allora la somma rimane pari altrimenti la somma diventa dispari inquanto si sta sommando 1 ad un numero pari.
2. Spostare di una posizione verso destra i bits di un numero binario corrispondere a dividere il numero per 2. Infatti prima dello scorrimento il generico bit di posizione *i* aveva peso 2^i mentre a seguito dello scorrimento lo stesso bit che ora occuperà la posizione *i-1* avrà peso 2^{i-1} (cioè peso dimezzato).
3. Spostare di una posizione verso sinistra i bits di un numero binario corrispondere a moltiplicare il numero per 2. Infatti prima dello scorrimento il generico bit di posizione *i* aveva peso 2^i mentre a seguito dello scorrimento lo stesso bit che ora occuperà la posizione *i+1* avrà peso 2^{i+1} (cioè peso doppio).
4. Un codice binario a *n* bits può rappresentare 2^n oggetti (per esempio i numeri interi compresi tra 0 e $2^n - 1$).

Da Ottale a Decimale

In un codice ottale, l'alfabero dei simboli utilizzato contiene solo 8 elementi: lo zero e sette (cioè $\mathcal{D} = \{0, 1, 2, 3, 4, 5, 6, 7\}$)

Ricordendo la Formula 1 e sapendo che *b=8* il numero in base 8 $N_8 = 6412$ in decimale vale:

$$\begin{aligned} N_{10} &= 6 \times 8^3 + 4 \times 8^2 + 1 \times 8^1 + 2 \times 8^0 = \\ &= 3072 + 256 + 8 + 2 = 3338 \end{aligned}$$

Esempi di Conversione da Decimale

La conversione da un numero decimale *N* ad un generico codice in base *b* si ottiene applicando il seguente semplice algoritmo:

1. Si ponga $N_0 = N$ e *i=1*
2. Si divida N_{i-1} per *b* e sia detto N_i il quoziente intero ed R_i il resto
3. Se $N_i = 0$ allora FINE
4. Si incrementi *i* di 1 (*i = i + 1*)
5. Ritorna al punto 2.

Alla fine dell'algoritmo il codice:

$$R_i R_{i-1} \ldots R_2 R_1$$

rappresenta la codifica in base *b* di *N*.

$$
\begin{array}{r|l}
N_0 = 74 & \\
N_1 = 37 & R_1 = 0 \\
N_2 = 18 & R_2 = 1 \\
N_3 = 9 & R_3 = 0 \\
N_4 = 4 & R_4 = 1 \\
N_5 = 2 & R_5 = 0 \\
N_6 = 1 & R_6 = 0 \\
N_7 = 0 & R_7 = 1
\end{array}
$$

Conversione da Decimale a Binario

Si voglia convertire il numero decimale 74 in binario.

Quindi in definitiva 74 in binario vale $R_7R_6R_5R_4R_3R_2R_1 = 1001010$.

Effettuiamo la verifica, cioè convertiamo il numero binario 1001010 in decimale, e vediamo se otteniamo 74:

$$
\begin{aligned}
1001010 &= 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = \\
&= 64 + 8 + 2 = 74
\end{aligned}
$$

Conversione da Ottale a Binario

Si voglia convertire il numero decimale 74 in ottale (cioè in base 8). Quindi in definitiva 74 in ottale

$$
\begin{array}{r|l}
N_0 = 74 & \\
N_1 = 9 & R_1 = 2 \\
N_2 = 1 & R_2 = 1 \\
N_3 = 0 & R_3 = 1
\end{array}
$$

vale $R_3R_2R_1 = 112$. Effettuiamo la verifica, cioè convertiamo il numero ottale 112 in decimale, e vediamo se otteniamo 74:

$$
\begin{aligned}
112 &= 1 \times 8^2 + 1 \times 8^1 + 2 \times 8^0 = \\
&= 64 + 8 + 2 = 74
\end{aligned}
$$

Numeri Interi Relativi

In un calcolatore i numeri interi relativi posso essere rappresentati in due modi:

- Rappresentazione in modulo e segno
- Rappresentazione in complemento a due

Nelle sottosezioni successive descriveremo in dettaglio ognuna delle due rappresentazioni.

Rappresentazione in Modulo e Segno

Nella rappresentazione in modulo e segno (da quì in avanti abbreviata con M&S), il segno di un numero intero relativo è codificato dal bit più significativo. Più precisamente dato un numero intero relativo N_{10} detto N_2 la rappresentazione binaria del numero intero naturale $|N_{10}|$, allora la rappresentazione in M&S di N_{10} si ottiene da N_2 aggiungendo come MSB un 1 se $N_{10} < 0$ e uno 0 se $N_{10} \geq 0$.

Esempio Si rappresenti il numero -27 in M&S utilizzando 8 bit.
Prima di tutto occorre convertire 27 in binario:

$$27 = 16 + 8 + 2 + 1 = 2^4 + 2^3 + 2^1 + 2^0 = 11011$$

Poichè si richiede che la codifica deve essere a 8 bit allora si procede in questo modo. Si aggiungono tanti zeri in testa alla codifica binaria di 27 in modo da portare la sua lunghezza a 7 bit.

$$27 = 0011011$$

Infine si aggiunge il bit di segno che nel nostro caso è 1 perchè -27 è un numero negativo. Quindi:

$$-27 = 10011011$$

Esempio Si rappresenti il numero 19 in M&S utilizzando 8 bit.
Prima di tutto occorre convertire 19 in binario:

$$19 = 16 + 2 + 1 = 2^3 + 2^1 + 2^0 = 1011$$

Poichè si richiede che la codifica deve essere a 8 bit allora si aggiungono tanti zeri in testa alla codifica binaria di 19 in modo da portare la sua lunghezza a 7 bit.

$$19 = 0001011$$

Infine si aggiunge il bit di segno che nel nostro caso è 0 perchè 19 è un numero positivo. Quindi:

$$19 = 00001011$$

Osservazione (range) Con una rappresentazione in M&S a *n* bits è possibile rappresentare tutti i numeri interi compresi tra $[-(2^{n-1} - 1), 2^{n-1} - 1]$. La dimostrazione è semplice. Infatti il più grande numero intero che può essere espresso si ottienen ponendo il MSB a 0 e tutti gli altri a 1 (cioè $2^{n-1} - 1$), mentre il più grande numero negativo che può essere espresso si ottiene ponendo tutti i bits a 1 (cioè $-(2^{n-1} - 1)$).

Osservazione (doppia rappresentazione dello 0) Nella rappresentazione in M&S si ha una doppia rappresentazione dello 0. Infatti, per esempio, i seguenti due numeri 1000 e 0000 rappresentati in M&S rappresentano entrambi il numero 0.

Rappresentazione in Complemento a Due

La rappresentazione dei numeri interi positivi in complemento a due (da quì in avanti abbreviata con C2) coincide con la rappresentazione binaria naturale (o pura). La rappresentazione di un numero intero negativo N_{10} in C2 si ottiene nel seguente modo. Detto N_2 la rappresentazione binaria del numero intero naturale $|N_{10}|$, si procede dal bit meno significativo di N_2 verso quello più significativo:

- se si incontrano tutti i bit 0, essi vengono lasciati inalterati;
- se si incontra il primo bit 1 anche esso viene lasciato inalterato;
- tutti i bit successivi al primo bit 1, vengono invertiti (0 diviene 1 e 1 diviene 0).

Esempi

- Il C2 di 10100 è 01100
- Il C2 di 01101001 è 10010111

Conversione da C2 a Decimale Sia un numero N_2 di *n* bits rappresentato in C2:

$$N_2 = d_{n-1}d_{n-2}\dots d_3d_2d_1d_0$$

Per converitire tale numero in decimale basta applicare la seguente formula:

$$N_{10} = -d_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} d_i \times 2^i \tag{2}$$

Cioè il MSB di un numero in C2 è pesato in modo negativo.

Esempio Converire il numero -29 in binario utilizzando la rappresentazione in C2 a 8 bits.

$$29 = 16 + 8 + 4 + 1 = 2^4 + 2^3 + 2^2 + 2^0 = 11101$$

Poichè è richiesto di utilizzare 8 bit si ha:

$$29 = 00011101$$

Quindi per ottenere la codifica di -29 basta fare il C2 di 00011101:

$$C2(00011101) = 11100011$$

Effettuiamo la verifica applicando la Formula 2:

$$N_{10} = -2^7 + 2^6 + 2^5 + 2^1 + 2^0 = -128 + 64 + 32 + 2 + 1 = -29$$

Osservazione Con una codifica in C2 a *n* bit si possono rappresentare tutti i numeri interi compresi nel rage $[-2^{n-1}, 2^{n-1} - 1]$. Infatti il più grande intero rappresentabile si ottiene ponendo il MSB a 0 e tutti gli altri *n-1* bits a 1 (cioè $2^{n-1} - 1$). Viceversa, il più grande numero negativo si ottiene ponendo il MSB a 1 e tutti gli altri a 0 (cioè 2^{n-1}).

Numeri Razionali

Esistono due possibili rappresentazioni per i numeri razionali:

- rappresentazione in virgola fissa,
- reppresentazione in virgola mobile.

Rappresentazione in Virgola Fissa

In un numero rappresentato in virgola fissa (da quì in avanti abbreviato con VF) a *n* bits, vengono utilizzati *I* bits per rappresentare la parte intera e *D* bits per rappresentare la parte decimale (ovviamente sarà *n* = *I* + *D*).

I-1	I-2	I-3	...	0	,	-1	-2	...	-D
intera						frazionaria			

Da VF a Decimale Per convertire un numero rappresentato in VF con *I* bits di parte intera e *D* bits di parte decimale:

$$N_2 = a_{I-1}a_{I-2}a_{I-3}\ldots a_1a_0, b_{-1}b_{-2}\ldots b_{-D}$$

basta applicare la seguente formula:

$$N_{10} = \sum_{i=0}^{I-1} a_i \times 2^i + \sum_{d=-1}^{-D} b_d \times 2^d \tag{3}$$

Esempio Convertire in decimale il numero binario in virgola fissa 1010,101.

- Parte Intera: $2^3 + 2^1 = 10$
- Parte Frazionaria: $2^{-1} + 2^{-3} = 0.5 + 0.125 = 0.625$

Quindi 1010,101 = 10,625.

Osservazione Nella rappresentazione in VF, la virgola è fissa e quindi è fisso il numero di bits assegnati per codificare la parte intera e quelli assegnati per codificare la parte frazionaria. Quindi se per esempio si fissa la virgola in modo tale che la gran parte dei bits è dedicata per codificare la parte intera, allora la precisione nel codificare numeri piccoli sarà molto bassa.

Da Decimale a VF La parte intera si codifica in binario. La parte frazionaria si converte così:

- si moltiplica per 2 e nel risultato si separa la parte intera;
- si ripete il procedimento fino a quando il risultato della moltiplicazione è 1.000... oppure si raggiunge la precisione richiesta.

Per esempio convertire il numero 5,625 nella rappresentazione in VF utilizzando 5 bit per la parte intera e 3 bit per la parte frazionaria (formato xxxx,xxx).

$$
\begin{array}{rl}
0{,}625 & \times \\
2 & \\
\hline
\mathbf{1}{,}250 & \times \\
2 & \\
\hline
\mathbf{0}{,}500 & \times \\
2 & \\
\hline
\mathbf{1}{,}000 &
\end{array}
$$

- Parte intera 5=0101
- Parte frazionaria

Per cui 5,635=0101,101.

Rappresentazione in Virgola Mobile

In un numero rappresentato in virgola mobile (da quì in avanti abbreviato con VM) vengono stabiliti un certo numero di bit assegnati per codificare la mantissa (*m*) ed un certo numero di bit per codificare l'esponente (*e*).

Tuttavia, a differenza degli interi, non esistono convenzioni adottate universalmente, e il lettore interessato dovrà sempre consultare il manuale del calcolatore per conoscere il tipo di rappresentazione adottato per la mantissa e per l'esponente.

A titolo di esempio si considere la seguente rappresentazione: In cui MSB è il segno mentre

31	30	29	...	24	23	,	22	21	...	1	0
segno	esponente				mantissa						

l'esponete è stato rappresentato con 7 bits e la mantissa con i restanti 24 bits.

- Il numero occupa in tutto 32 bit.
- La mantissa è rappresentata in modulo (bits 0-23) e segno (bit 31).
- L'esponente è rappresentato nei bit 1-8 in *eccesso 64* cioè, il vero esponente si ottiene dalla sua rappresentazione sottraendo 64. Ad esempio:
  - 0000000 corrisponde ad un esponente -64 (perchè 0-64=-64)
  - 1111111 corrisponde ad un esponente +63 (perchè 127-64=63)

Un numero in VM è rappresentato come:

$$m \times 10^e$$

Esempi di Conversione in VM Si converta il numero numero N=18.75 nella rappresentazione in VM. Si procede nel seguente modo:

1. si normalizza il numero in modo che la parte intera si 0. Nel nostro caso occorre spostare la virgola di 2 posizioni verso sinistra quindi si ha:

- N=0.1875
- e=+2 (perchè la virgola è stata spostata di due posizioni, e + perchè questo spostamento è avvenuto verso sinistra)

2. si converte il numero ottenuto descritto nel paragrafo utilizzando la rappresentazione a VF in cui solo 1 bit è utilizzato per codificare la parte intera mentre tutti gli altri sono utilizzati per codificare la parte razionale.. Nel nostro caso: Quindi 0.1875=0011

$$\begin{array}{rl} 0{,}1875 & \text{x} \\ 2 & \\ \hline \mathbf{0}{,}3750 & \text{x} \\ 2 & \\ \hline \mathbf{0}{,}7500 & \text{x} \\ 2 & \\ \hline \mathbf{1}{,}5000 & \text{x} \\ 2 & \\ \hline \mathbf{1}{,}0000 & \text{x} \end{array}$$

3. Si codifichi in binario l'esponente. Ricordando che l'esponente è in eccesso 64 deve essere:

$$e - 64 = 2 \Rightarrow e = 66$$

La sua rappresentazione in binario è:

$$66 = 64 + 2 = 2^6 + 2^1 = 1000010$$

4. In definitiva si ha:

$$18.75 = 0100001000011000000...000$$

dove il primo bit (0) è il bit di segno, i successivi 7 bit codificano l'esponente in eccesso 64, la restante parte codifica la mantissa in VF in cui il primo bit (0) è la parte intera, e la restante parte codifica la parte razionale.

Range dei Valori Rappresentabili Il più piccolo numero diverso da zero rappresentabile è (in valore assoluto)

$$\frac{1}{2^{bitspermantissa-1}} \times 10^{-64}$$

Infatti l'esponente più negativo è -64, e la mantissa più piccola è quella ottenuta con una codifica in cui solo il LSB vale 1 mentre tutti gli altri bit sono 0.

Il più grande numero rappresentabile sarà, al contrario, quello avente la massima mantissa e il massimo esponente cioè:

$$1 \times 10^{63}$$

Infatti la mantissa più grande è quella che codifica 1 e l'esponente più grande è 63.

Quindi si capisce com'è possibile con questa rappresentazione, rappresentare sia numeri molto piccoli che numeri molto grandi.