Appunti sulla codifica binaria

1 Notazione Posizionale

Il sistema di numerazione utilizzato correntemente è un sistema di numerazione posizionale, ossia un sistema in cui i simboli (detti cifre) usati per esprimere i numeri assumono valori diversi in funzione della posizione che occupano.

Ad esempio se consideriamo il numero 545, il 5 più a sinistra assume valore cinquecento, il 4 assume valore quaranta mentre il 5 più a destra assume valore cinque. Dunque lo stesso simbolo (in questo caso il 5) assume valore diversi a seconda della posizione.

I sistemi di numerazione posizionale necessitano del numero 0 per rappresentare le posizioni vuote, ad esempio per far assumere al simbolo 3 il valore di trecento è necessario fargli occupare la terza posizione e quindi inserire due volte il simbolo 0, ottenendo così il numero 300.

La notazione posizionale permette, inoltre, di utilizzare un ridotto numero di simboli per rappresentare numeri di qualsiasi grandezza. In particolare, è sufficiente scegliere un certo numero di simboli distinti (ossia di cifre) maggiore di uno, per rappresentare qualsiasi numero. Il numero di simboli definisce la base del sistema di numerazione adottato.

Il sistema correntemente utilizzato è in base dieci, sono quindi necessari dieci simboli distinti, $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, per rappresentare qualsivoglia numero.

Un numero a n cifre $C_{n-1}, C_{n-2}, \ldots, C_1, C_0$, espresso genericamente in base β, dove C_i rappresenta la cifra in posizione i, si può scrivere in formula, in accordo alla notazione posizionale, come segue:

$$C_{n-1} \cdot \beta^{n-1} + C_{n-2} \cdot \beta^{n-2} + \cdots + C_1 \cdot \beta^1 + C_0 \cdot \beta^0.$$

Consideriamo, di nuovo il numero **545** (quindi $n = 3$, $C_2 = 5$, $C_1 = 4$ e $C_0 = 5$), questo può essere espresso come: $\mathbf{5} \cdot 10^2 + \mathbf{4} \cdot 10 + \mathbf{5} \cdot 1$.

È inoltre importante sottolineare come gli algoritmi adottati per effettuare i calcoli siano indipendenti dalla base scelta, ossia, qualsiasi sia la base, è possibile utilizzare la stessa metodologia che si usa nel sistema decimale per effettuare somme, sottrazioni, moltiplicazioni, divisioni, elevamento a potenza, etc..

1.1 Intervalli di rappresentazione

È noto che l'insieme dei numeri naturali ha cardinalità infinita e quindi esistono numeri composti da un insieme infinito di cifre. Tuttavia, se i numeri devono essere elaborati ad esempio da uno strumento elettronico di calcolo, vi sono delle limitazioni fisiche al numero di cifre massimo che possono costituire un numero.

In base, quindi, al dispositivo considerato si ha un diverso intervallo di numeri che è possibile elaborare.

Si consideri ad esempio il display di una calcolatrice a 12 cifre, questo può contenere solo numeri compresi tra 0 e 999˙999˙999˙999.

Si noti che è possibile rappresentare numeri costituiti da un numero di cifre inferiore a n utilizzando esattamente n cifre, completando il numero inserendo degli zeri a sinistra, operazione che ovviamente non modifica il valore del numero.

Ad esempio, è possibile rappresentare su 10 cifre il numero 25, inserendo otto zeri a sinistra: 0000000025.

Fissato il numero di cifre, è immediato calcolare l'intervallo di numeri rappresentabili. In particolare, se n è il numero di cifre e β è la base, è possibile rappresentare tutti i numeri nell'intervallo $[0, (\beta^{n-1})]$.

Ad esempio, su 12 cifre decimali è possibile rappresentare tutti i numeri compresi tra 0 e $(10^{12} - 1) = 999˙999˙999˙999$.

1.2 Moltiplicazioni e Divisioni per potenze della base

La notazione posizionale implica anche la corrispondenza tra le operazioni di moltiplicazione e divisione per potenze della base e spostamenti a sinistra e a destra delle cifre. In particolare, dato un numero $C_{n-1} \dots C_1 C_0$ a n cifre in base β, il prodotto del numero per l'i-esima potenza della base corrisponde ad uno spostamento a sinistra di i posizioni delle cifre con il conseguente inserimento a destra di i zeri, in formula:

$$C_{n-1} \dots C_1 C_0 \cdot \beta^i = C_{n-1} \dots C_1 C_0 \underbrace{00...0}_{i}.$$

Ad esempio, il prodotto tra il numero decimale 324 e 10^2 equivale allo spostamento a sinistra di 2 posizioni delle cifre costituenti il numero ed il conseguente inserimento di due zeri, ottenendo così il numero 32400.

Analogamente, dato un numero $C_{n-1} \dots C_1 C_0$ a n cifre in base β, il quoziente della divisione tra il numero e l'i-esima potenza della base corrisponde ad uno spostamento a destra di i posizioni delle cifre, ossia al troncamento delle ultime i cifre.

Ad esempio, il quoziente della divisione tra il numero decimale 324 e 10^2 equivale allo spostamento a destra di 2 posizioni delle cifre costituenti il numero ed il conseguente troncamento delle ultime due cifre, ottenendo così il numero 3.

2 Rappresentazione di interi in base 2

Come detto nel capitolo precedente, un qualsiasi numero di simboli (cifre) ci permette di esprimere qualsiasi numero. A parte il sistema in base 10 detto anche decimale che è quello correntemente usato, tanti altri sistemi di numerazione sono o sono stati utilizzati, ad esempio sistemi in base 8 (otto simboli distinti), sistemi in base 12 (dodici simboli distinti), sistemi in base 16 (sedici simboli distinti $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$), sistemi in base 60 (o sessagesimali) (sessanta simboli distinti), e tanti altri.

Di particolare interesse, è il sistema in base 2 (o binario) che necessita di soli due simboli cioè il minimo numero possibile di simboli.

Nel sistema in base 2, quindi, ogni cifra può assumere solo due possibili valori 0 e 1.

Si consideri ad esempio il numero binario **11001**, in accordo alla notazione posizionale questo equivale a:

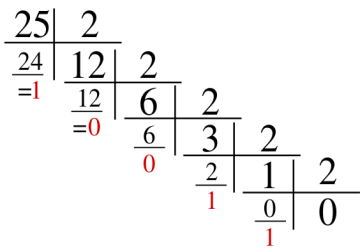
$$\mathbf{1} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{0} \cdot 2 + \mathbf{1}$$

quindi al numero decimale 25. Una singola cifra del sistema binario prende il nome di *bit*, mentre una sequenza di 8 cifre binarie prende il nome di *byte*. Sono poi utilizzati vari multipli del byte tra cui: il *kilobyte* equivalente a 1024 byte, il *megabyte* equivalente a 1024 kilobyte, il *gigabyte* equivalente a 1024 megabyte e il *terabyte* equivalente a 1024 gigabyte.

2.1 Conversione da decimale a binario

Per trasformare un numero dalla base 10 alla base 2 è necessario effettuare una sequenza di divisioni per 2 e tenere traccia dei quozienti e dei resti. In particolare, si effettua una divisione per 2, si conserva il resto e si considera il quoziente. Il quoziente viene diviso nuovamente per 2 e ancora una volta si conserva il resto e si considera il quoziente, e così via fin quando il quoziente non assume valore 0. Il numero in base 2 si ottiene considerando i resti in ordine inverso.

Si consideri ad esempio il numero decimale 25, per ottenerne la rappresentazione in base 2 si procede come in figura:



Leggendo i resti in ordine inverso, si ottiene 11001 che rappresenta appunto il numero 25.

Come detto, nella rappresentazione dei numeri, il numero massimo di cifre di un numero è spesso fissato ed i numeri composti da un numero di cifre inferiori vengono completati inserendo degli zeri a sinistra.

Ad esempio, assumendo che il numero di cifre massimo considerato sia 8, il numero 25 verrebbe scritto in binario su otto cifre come 00011001.

Nel seguito della trattazione, assumeremo per semplicità che il numero di cifre a disposizione sia sempre 8. Si noti che l'intervallo dei numeri rappresentabili in 8 cifre binarie è $[0, (2^8 - 1)]$, cioè $[0, 255]$.

2.2 Numeri negativi

Per rappresentare un qualsiasi numero naturale, come detto, occorre un numero di simboli distinti pari alla base. Ma cosa accade se si vuole rappresentare un qualsiasi numero intero e, quindi, anche un numero negativo? Per rappresentare i numeri negativi, nel sistema decimale viene correntemente utilizzato un

ulteriore simbolo, il "−", che posto davanti al numero indica che si tratta di un numero negativo.

Lo stesso metodo si potrebbe, teoricamente, utilizzare in qualsiasi sistema di numerazione, quindi prevedere anche in base 2 la presenza dell'ulteriore simbolo "−" per rappresentare i numeri negativi.

Tuttavia, in informatica, vi è la necessità di non utilizzare più di due simboli per rappresentare qualsivoglia informazione[1]. Di conseguenza è indispensabile adottare un meccanismo per rappresentare i numeri negativi attraverso l'utilizzo dei soli due simboli 0 e 1.

2.2.1 Rappresentazione Modulo e Segno

Una possibile via, nonché la più semplice, per rappresentare i numeri negativi consiste nel considerare il primo bit di un numero (cioè quello più a sinistra) non come cifra del numero ma come segno. In particolare, se il bit più a sinistra del numero vale 0 allora il numero si considera positivo, se vale 1 si considera negativo. Se si considera nuovamente il numero 25 (espresso come 00011001 in binario), questo verrebbe rappresentato, come segue:

$$00011001 \to +25, \qquad 10011001 \to -25.$$

Quindi, il più grande numero rappresentabile in 8 bit diventa $01111111 \to +127$, mentre il più piccolo numero rappresentabile in 8 bit diventa $11111111 \to -127$.

Questa rappresentazione è molto semplice e di facile implementazione, tuttavia presenta due problemi rilevanti.

Rappresentazione dello 0. La rappresentazione *Modulo e Segno* prevede due modalità distinte per rappresentare il numero 0, in particolare sia 00000000 sia 10000000 rappresentano il numero 0 dato che in questo caso il segno non ha alcuna rilevanza.

Correttezza delle operazioni. L'algoritmo di addizione fallisce in presenza di numeri negativi. Si consideri ad esempio la somma $(+25)+(-25)$. Il risultato atteso di quest'operazione è 0. Verifichiamo applicando l'algoritmo:

$$\begin{array}{r} {\scriptstyle 1\,1\qquad 1} \\ 00011001+ \\ 10011001= \\ \hline 10110010 \end{array}$$

Il risultato ottenuto è 10110010 corrispondente al numero decimale −50 che, ovviamente, non è il risultato corretto.

[1] Questa necessità è dovuta alle problematiche insite nell'elaborazione dell'informazione attraverso strumenti elettronici.

2.2.2 Rappresentazione Complemento a Due

Per ovviare ai problemi della rappresentazione *Modulo e Segno* è stata introdotta la rappresentazione *Complemento a Due*.

Tale rappresentazione prevede la costruzione di un numero negativo come segue: si considera il valore assoluto del numero, di questo si calcola il *complemento a uno*, ossia si invertono tutti i bit ($0 \to 1, 1 \to 0$), infine si somma 1 al risultato.

Ad esempio, il numero -25 verrebbe scritto in *Complemento a Due*, come segue: si parte dal valore assoluto di -25 che in binario è 00011001

- il complemento a uno di 00011001 è 11100110;
- sommando 1 si ottiene 11100111.

Il numero -25 equivale quindi a 11100111.

Per la rappresentazione in *Complemento a Due* vale la proprietà secondo la quale il primo bit del numero (quello più a sinistra) indica il segno: se vale 1 il numero è negativo se vale 0 il numero è positivo.

La metodologia presentata precedentemente è ugualmente utilizzabile per convertire un numero dalla rappresentazione in *Complemento a Due* alla rappresentazione decimale.

Ad esempio si consideri il numero 11100110. Essendo il primo bit pari a 1, si tratta di un numero negativo.

- Il complemento a uno di 11100110 vale 00011001.
- Sommando 1 si ottiene 00011001.

Il valore assoluto del numero è quindi 00011001 che in decimale corrisponde a 26. Il numero cercato è quindi -26.

Verifichiamo ora come questa rappresentazione risolva i problemi della rappresentazione *Modulo e Segno*.

Rappresentazione dello 0. In *Complemento a Due* la rappresentazione dello 0 è univoca e corrisponde a 00000000. Consideriamo, infatti, il numero 10000000: il suo complemento a uno vale 01111111, sommando 1 si ottiene 10000000 che considerato come valore assoluto (ossia senza segno) vale 128, pertanto il numero 10000000 equivale a -128.

Correttezza delle operazioni. Con la rappresentazione *Complemento a Due* viene garantita la correttezza delle operazioni. Consideriamo nuovamente l'operazione $(+25) + (-25)$.

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ \ } \\ 00011001+ \\ 11100111= \\ \hline 00000000 \end{array}$$

Il risultato ottenuto è 00000000 corrispondente al numero decimale 0 che, ovviamente, è il risultato corretto. Si noti che l'ultimo riporto (al nono bit) viene ignorato perché stiamo considerando numeri a otto bit.

Per quanto riguarda l'intervallo di rappresentabilità, il più piccolo numero rappresentabile in n bit è costituito da un 1 iniziale seguito da $n-1$ bit pari a 0, mentre il più grande numero rappresentabile è costituito da un 0 iniziale seguito da $n-1$ bit pari a 1. In formula, l'intervallo è esprimibile come $[-2^{n-1}, 2^{n-1}-1]$. Quindi, nel caso ad esempio di numeri a 8 bit, l'intervallo di rappresentabilità è $[-128, 127]$.

Per maggiore chiarezza, di seguito vengono presentati tutti i numeri esprimibili in 4 bit:

$$\begin{array}{ll} 0111 \to 7 & 1111 \to -1 \\ 0110 \to 6 & 1110 \to -2 \\ 0101 \to 5 & 1101 \to -3 \\ 0100 \to 4 & 1100 \to -4 \\ 0011 \to 3 & 1011 \to -5 \\ 0010 \to 2 & 1010 \to -6 \\ 0001 \to 1 & 1001 \to -7 \\ 0000 \to 0 & 1000 \to -8 \end{array}$$

Si noti come l'intervallo di rappresentabilità sia $[-2^3, 2^3 - 1] = [-8, 7]$.

2.3 Moltiplicazioni e Divisione per potenze di 2

Riprendendo il concetto espresso nel Capitolo 1.2, dato un numero in base 2, moltiplicarlo per 2^i equivale a spostare a sinistra di i posizioni le cifre del numero così come dividerlo per 2^i (o, analogamente, moltiplicarlo per 2^{-i}) equivale a spostare a destra di i posizioni le cifre del numero.

Ad esempio, si consideri l'operazione $18 \cdot 8 = 144$. Il 18 in binario equivale a 10010, quindi l'operazione equivale a moltiplicare 10010 per la terza potenza della base ($8 = 2^3$). Spostando il numero a sinistra ed inserendo di conseguenze tre zeri si ottiene 10010000 che in decimale corrisponde esattamente a 144.

Analogamente, si consideri l'operazione $43/8 = 43/2^3 = 43 \cdot 2^{-3} = 5$. Il 43 in binario equivale a 101011, quindi l'operazione equivale a dividere 101011 per la terza potenza della base ($8 = 2^3$). Spostando il numero a destra e troncando di conseguenza le ultime tre cifre si ottiene 101 che in decimale corrisponde esattamente a 5.

3 Rappresentazione di numeri reali

I paragrafi precedenti hanno trattato la codifica binaria di numeri interi positivi e negativi. In questo paragrafo verrà, invece, analizzata la codifica binaria di numeri reali.

Un numero reale è costituito da una parte intera e da una parte frazionaria.

Ad esempio, nel numero 753,1683 la parte intera è 753 mentre 1683 è la parte frazionaria.

Analogamente al caso intero, anche per i numeri reali vige la notazione posizionale, dove per la parte frazionaria l'esponente assume valori negativi. Un

numero reale in base β, composto da una parte intera a n cifre e una parte frazionaria a m cifre,

$$C_{n-1} \dots C_1 C_0, D_1 \dots D_m$$

può essere scritto come

$$C_{n-1} \cdot \beta^{n-1} + \dots + C_1 \cdot \beta + C_0 + D_1 \cdot \beta^{-1} + \dots + D_m \cdot \beta^{-m}.$$

Quindi, il numero decimale 753,1683 ($n = 3$ e $m = 4$) può essere riscritto come:

$$7 \cdot 10^2 + 5 \cdot 10 + 3 + 1 \cdot 10^{-1} + 6 \cdot 10^{-2} + 8 \cdot 10^{-3} + 3 \cdot 10^{-4}$$

Analogamente, il numero binario 101,1001 può essere scritto come:

$$1 \cdot 2^2 + 0 \cdot 2 + 1 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4}$$

equivalente al numero decimale 5,5625.

Formalmente, i numeri che verranno considerati sono numeri razionali e per approssimazione i numeri reali in quanto, nella pratica, non è possibile elaborare numeri con un infinito numero di cifre frazionarie.

Vi sono due principali approcci per codificare un numero razionale: *virgola fissa* e *virgola mobile*.

3.1 Formato a Virgola Fissa

Secondo questo approccio, viene fissato a priori il numero di cifre che costituisce la parte intera del numero (espresso in qualsivoglia base) ed il numero di cifre che costituisce la parte frazionaria espressa nella stessa base della parte intera.

Questo approccio è però poco flessibile nella rappresentazione dei numeri in quanto non permette all'utilizzatore di scegliere quante cifre frazionarie memorizzare in accordo alle sue esigenze.

Per capire meglio la questione, si consideri il seguente esempio. Supponiamo di avere a disposizione 12 cifre decimali e di voler utilizzare un formato a virgola fissa con 6 cifre per la parte intera e 6 cifre per la parte frazionaria. In questo caso è possibile rappresentare solo numeri la cui parte intera occupi al più 6 cifre e la cui parte frazionaria occupi analogamente al più 6 cifre. Non è pertanto possibile rappresentare ad esempio il numero $1\dot{2}35\dot{1}62{,}5$ sebbene occupi complessivamente 8 cifre (7 per la parte intera e 1 per la parte frazionaria) e si abbiano a disposizione 12 cifre. Analogamente, non è possibile memorizzare il numero 0,12035432 sebbene occupi complessivamente 9 cifre (1 per la parte intera e 8 per la parte frazionaria) e si abbiano a disposizione 12 cifre.

3.2 Formato a Virgola Mobile

In accordo a questa rappresentazione, il numero di cifre destinato alla parte intera ed alla parte frazionaria non è stabilito a priori ma può essere definito di volta in volta a seconda delle esigenze. Questo tipo di rappresentazione è molto simile alla notazione scientifica.

Un numero in virgola mobile è composto da due campi, la *mantissa* (in seguito indicata con M) e l'*esponente* (in seguito indicato con e), tali che il numero possa essere scritto come $\pm M \cdot \beta^{\pm e}$, dove β è la base in cui è rappresentato il

numero. La forma normalizzata della mantissa prevede che la parte intera sia sempre 0 e la cifra più significativa della parte frazionaria sia maggiore di 0.

Quindi, ad esempio il numero (in base 10) 753,168 può essere riscritto come $0{,}753168 \cdot 10^3$.

La rappresentazione in virgola mobile prevede, quindi, la memorizzazione di due interi: l'uno rappresentante la mantissa (nell'esempio l'intero 753168) e l'altro rappresentante l'esponente (nell'esempio l'intero 3). A seconda del valore assunto dall'esponente, è possibile stabilire quante cifre della mantissa dedicare alla parte intera e quante alla parte frazionaria.

Supponiamo ad esempio di avere a disposizione 5 cifre decimali e che ne vengano dedicate 3 alla mantissa e 2 all'esponente. Il più grande numero che è possibile memorizzare è il numero $0{,}999 \cdot 10^{99}$ mentre il più piccolo numero positivo è $0{,}1 \cdot 10^{-99}$. Si noti come nel primo caso il numero non contenga parte frazionaria, mentre nel secondo caso non contenga parte intera.

Per evidenziare un ulteriore confronto tra i due formati, si considerino gli intervalli di rappresentabilità nel caso si abbiano 5 cifre decimali a disposizione e si utilizzino 3 cifre per la parte intera e 2 cifre per la parte frazionaria nel caso di numero in virgola fissa, e si utilizzino 3 cifre per la mantissa e 2 per l'esponente nel caso di numero in virgola mobile. Esprimendo il numero in virgola fissa il più piccolo numero maggiore di zero rappresentabile è 0,01 mentre il più grande numero positivo rappresentabile è 999,99. Viceversa, esprimendo il numero in virgola mobile il più piccolo numero maggiore di zero rappresentabile è $0{,}1 \cdot 10^{-99}$ mentre il più grande numero positivo rappresentabile è $0{,}999 \cdot 10^{99}$.

3.3 Rappresentazione standard di numeri reali in base 2

Gli standard per la rappresentazione in binario dei numeri razionali sono due: il formato a 32 bit ed il formato a 64 bit. Il primo viene utilizzato per il tipo di dati noto come *float*, il secondo per il tipo di dati noto come *double*.

Rispetto a quanto detto nel paragrafo precedente, vi sono delle piccole differenze nella rappresentazione standard di numeri reali binari.

1. La forma normalizzata della mantissa prevede che la parte intera sia 1. Quindi, il numero 101,1001 viene riscritto come $1{,}011001 \cdot 2^2$ e non come $0{,}1011001 \cdot 2^3$.

2. La mantissa viene memorizzata utilizzando la notazione *Modulo e Segno*.

Standard per numeri a precisione singola. Nel caso di numeri a precisione singola, i 32 bit sono così suddivisi:

- 1 bit per il segno della mantissa;
- 8 bit per l'esponente;
- 23 bit per la mantissa.

L'esponente viene memorizzato come un numero positivo compreso tra 0 e 255.

Il valori estremi, ossia 0 e 255, hanno un significato speciale quindi il valore del campo esponente è compreso, in realtà, tra 1 e 254. Al valore di questo

campo viene sottratto 127 ottenendo così un numero compreso tra -126 e 127 che è il reale valore dell'esponente.

Il significato dei valori 0 e 255 per l'esponente è riassunto nel seguente schema:

Significato	*Esponente*	*Mantissa*
Zero	0	0
Numero Non-Normalizzato	0	$\neq 0$
Numero Normalizzato	1..254	*qualsiasi*
Infinito	255	0
NaN	255	$\neq 0$

NaN è l'acronimo di *Not-A-Number* e si usa per rappresentare un numero non valido, come ad esempio il risultato dell'operazione $0/0$ o la radice quadrata di un numero negativo.

Si consideri, ad esempio, il numero 5,5625. Questo viene rappresentato in forma normalizzata come $1{,}011001 \cdot 2^2$. In accordo allo standard esposto, il segno vale 0, la mantissa vale 011001 e l'esponente vale 129 (ossia $127 + 2$).

0	10000001	01100100000000000000000

Si noti come per completare la mantissa ed esprimerla in 23 bit sia necessario aggiungere 17 bit pari a 0 a destra[2].

Standard per numeri a doppia precisione. Nel caso di numeri a doppia precisione, i 64 bit sono così suddivisi:

- 1 bit per il segno della mantissa;
- 11 bit per l'esponente;
- 52 bit per la mantissa.

La rappresentazione è del tutto analoga al caso precedente, ad eccezione dell'esponente per il quale i valori 0 e 2047 sono riservati, i valori compresi tra 1 e 2046 sono utilizzabili per rappresentare numeri regolari e, infine, dal valore del campo esponente bisogna sottrarre 1023 per ottenere il valore reale dell'esponente.

[2]Per conferma della validità dell'operazione si consideri il numero decimale 5,5625, volendo esprimere la parte frazionaria su 6 cifre è sufficiente aggiungere due zeri a destra, ottenendo il numero equivalente $5,562500$.