Università di Trieste

Laurea in ingegneria elettronica e informatica

Enrico Piccin - Corso di Reti Logiche - Prof. Stefano Marsi

Anno Accademico 2022/2023 - 6 Ottobre 2022

Indice

1 Sistemi di numerazione e codici **2**
1.1 Conversione tra basi diverse di numeri interi 2
1.2 Conversione tra basi diverse di numeri frazionari 2
1.3 Aritmetica binaria 3
1.4 Rappresentazione dei numeri negativi 4
1.5 Errori nei risultati 4
1.6 Moltiplicazione e Divisione 5
1.7 Casting 6
1.8 Codici 7

6 Ottobre 2022

1 Sistemi di numerazione e codici

Di seguito si espone la definizione di **sistema di numerazione**:

SISTEMA DI NUMERAZIONE

Un **sistema di numerazione** è un **insieme di simboli** (cifre) e regole, le quali consentono di associare ad una stringa di cifre il corrispondente valore numerico.
I codici decimale, binario, ottale o esadecimale sono tutti codici posizionali, il cui valore dipende dalla posizione delle cifre.

Osservazione: La base 2 è la più piccola possibile, in cui i bit sono associati agli stati ON/OFF. Le basi 8 e 16, invece, permettono rappresentazioni più compatte del numero binari, soprattutto perché il passaggio da base 2 a base 8 o 16 e viceversa è particolarmente facile

$$55_10 = 110111_2 \tag{1}$$
$$110111_2 = 37_16 = 67_8 \tag{2}$$

1.1 Conversione tra basi diverse di numeri interi

La conversione da base 10 a base 2, prevede di adottare il metodo delle **divisioni successive**: si divide ripetutamente il numero per la base voluta fino ad ottenere un quoziente nullo e si memorizzano i resti (la seq. dei resti ordinata rappresenta la notazione).
Per quanto detto, il passaggio da basi B a B^n e viceversa risulta particolarmente semplice:

$$157_10 = 10011101_2 = 235_8 = 9D_16$$

Osservazione: Si osservi che per convertire un numero da base 2 a base 10, non solo è possibile usare le potenze del due, ma è anche possibile partire dal bit più significativo e moltiplicarlo per 2, sommarlo al bit successivo e moltiplicare per 2, e via dicendo fino ad esaurire tutti i bit.

1.2 Conversione tra basi diverse di numeri frazionari

Com'è noto, la virgola distingue le cifre che vanno moltiplicate per la base B con esponente positivo da quelle con esponente negativo, per cui

$$0.101 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 0.5 + 0.125 = 0.625$$

Oppure si può anche traslare di 3 posizioni la virgola (che in binario vuole dire moltiplicare per $2^3 = 8$) e convertire il numero binario come se fosse intero e poi dividerlo ancora per $2^3 = 8$, per cui

$$0.101 \quad \rightarrow \quad 0.101 \cdot 2^3 = 101 = 5 \quad \rightarrow \quad 0.101 = \frac{5}{2^3} = 0.625$$

Se, invece, bisogna passare da decimale a binario, si deve procedere per moltiplicazioni successive:

$$0.375_10 \quad \rightarrow \quad 0.375 \cdot 2 = 0.750 \quad \rightarrow \quad \mathbf{0} + 0.750$$
$$0.750_10 \quad \rightarrow \quad 0.750 \cdot 2 = 1.500 \quad \rightarrow \quad \mathbf{1} + 0.500$$
$$0.500_10 \quad \rightarrow \quad 0.500 \cdot 2 = 1.000 \quad \rightarrow \quad \mathbf{1} + 0.000$$

Ecco, quindi, che il valore binario è stato ottenuto:

$$0.375_10 = 0.011_2$$

Osservazione: Ovviamente, è possibile che il processo sopra descritto cada in una ripetizione periodica. Allora, il processo di approssimazione può avvenire secondo due modalità:

1. Per **troncamento**, in cui si lascia semplicemente il valore binario ottenuto così com'è;
2. Per **arrotondamento**, in cui si considera il bit immediatamente successivo all'ultimo di quelli che si sta considerando e lo si somma all'ultima cifra, come mostrato di seguito:

$$011011|\boxed{1}01101 \rightarrow 011011 + \boxed{1} = 011100$$

Non sorprende, poi, osservare che se con una base una notazione frazionaria richiede un numero finito di cifre, potrebbe richiederne infinite con una diversa notazione.

1.3 Aritmetica binaria

L'addizione binaria è molto semplice, mentre la sottrazione risulterebbe particolarmente ostica, a meno che non si considerasse la complementazione. Infatti, dovendo eseguire, in decimale, la differenza $123 - 73$, è sufficiente eseguire la somma $123 + \text{comp}_{10}(73)$, in cui $\text{comp}_{10}(73)$ si calcola come segue

$$\begin{array}{ccc} 9 & 9 & 9 \; - \\ & 7 & 3 \; + \\ & & 1 \; = \\ \hline 9 & 2 & 7 \end{array}$$

per cui si ottiene $123 + \text{comp}_{10}(73) = 123 + 927 = 1|050 \rightarrow 50$, eliminando l'1 del migliaio, in quanto aggiunto prima per la complementazione. Analogamente in binario.

Osservazione: Si osservi che, per ogni base B, esistono due complementi per un numero N:

- Complemento a B, definito come $C_B = B^n - N$
- Complemento a $B - 1$, definito come $C_{B-1} = B^n - 1 - N$

Non solo, ma dalla differenza di N_1 ed N_2, vi possono essere due casi:

- $N_1 \geq N_2$: il risultato risulta maggiore o uguale a B^n, che pertanto va eliminato dal risultato finale (eliminazione dell'1 più significativo oltre il range del numero stesso)
- $N_1 < N_2$: il risultato risulta minore di B^n, e deve essere inteso come complemento a B (pertanto rappresentante di un numero negativo) del risultato. Per conoscerne il valore assoluto, è necessario ri-complementarlo.

Si consideri, infatti, l'esempio seguente, in cui si esegue la differenza $21 - 46$, ovvero:

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

In cui non è stato ottenuto un 1 nell'ultima operazione finale, pertanto il risultato 110111 deve essere ulteriormente complementato, ottenendo $011001_2 = 25$, che è il valore assoluto della differenza.

1.4 Rappresentazione dei numeri negativi

I numeri negativi possono pertanto essere rappresentati in base al loro complemento a B. In base $B = 10$, ciò non risulta essere usuale, ma si preferisce impiegare un segno grafico $-$.
In binario, ciò non risulta essere possibile, in cui i numeri negativi vengono rappresentati in base al loro complemento a 2, usando il bit più significativo viene impiegato come **bit di segno**:

in cui la convenzione sul bit di segno è

- **0**: numero positivo
- **1**: numero negativo

Attenzione che, eliminato il bit di segno in un numero binario

- nel caso di un numero positivo (quindi avendo eliminato il bit 0), i restanti numeri rappresentano il numero stesso:
$$01001_2 = +9_{10}$$
- nel caso di un numero negativo (quindi avendo eliminato il bit 1), i restanti numeri rappresentano il numero complementato:
$$11001_2 = -C_2(1001) = -0111_2 = -7_{10}$$

1.5 Errori nei risultati

Il risultato di un'operazione somma/sottrazione è coerente solo se il risultato non esce dal range dei numeri rappresentabili, per cui

- il risultato è **corretto** se
  - non si è avuto alcun riporto, nè nel bit di segno nè fuori dalla parola;
  - si sono avuti riporti in entrambi;
- il risultato è **errato** se si è avuto un solo riporto, o sul segno, o fuori dalla parola;

Dal punto di vista circuitale, per determinare se si è ottenuto un risultato corretto o meno, sarà sufficiente considerare il bit di riporto sul segno e quello fuori dalla parola e porli in XOR: se lo XOR produce come uscita 1, allora si è verificato un errore, altrimenti il risultato è corretto.

7 Ottobre 2022

Non deve sorprendere che un numero in una base non risulta essere periodico, mentre in altre basi esso lo è, in quanto le frazioni a disposizione sono differenti a seconda della base stessa; per esempio, le frazioni a disposizione per la base 3 sono

$$3^{-1} = \frac{1}{3}, 3^{-2} = \frac{1}{9}, 3^{-3} = \frac{1}{27}$$

ed ecco che quindi $\frac{1}{3}$, in base 3, si rappresenta come $0,1_3$.
Inoltre, se si adotta una notazione **unsigned** su n bit, i possibili numeri rappresentabili sono 2^n, da 0 a $2^n - 1$. Invece, se si adotta una notazione **signed** su n bit, i numeri rappresentabili sono i valori da 0 a $2^{n-1} - 1$, e da -1 a -2^{n-1}; tuttavia l'intervallo è il medesimo.
Nella tabella seguente si espongono i valori interi **signed** su 4 bit e i corrispondenti valori decimali se si considera la notazione con la virgola **signed** su 4 bit (utilizzando 2 bit dopo la virgola), ottenendo lo schema seguente:

Intero					Virgola
7	0	1	1	1	1.75
6	0	1	1	0	1.50
5	0	1	0	1	1.25
4	0	1	0	0	1.00
3	0	0	1	1	0.75
2	0	0	1	0	0.50
1	0	0	0	1	0.25
0	0	0	0	0	0.00
-1	1	1	1	1	-0.25
-2	1	1	1	0	-0.50
-3	1	1	0	1	-0.75
-4	1	1	0	0	-1.00
-5	1	0	1	1	-1.25
-6	1	0	1	0	-1.50
-7	1	0	0	1	-1.75
-8	1	0	0	0	-2.00

1.6 Moltiplicazione e Divisione

La moltiplicazione binaria è molto semplice, e segue la regola seguente:

- $0 \cdot 0 = 0$
- $1 \cdot 0 = 0$
- $0 \cdot 1 = 0$
- $1 \cdot 1 = 1$

e si basa sull'automatismo dello **shift & add**, come mostrato nel seguito:

$$\begin{array}{cccc} & 1 & 1 & 0 & \times \\ & & 1 & 0 & = \\ \hline & 0 & 0 & 0 & + \\ 1 & 1 & 0 & & = \\ \hline 1 & 1 & 0 & 0 & \end{array}$$

e la divisione viene eseguita, di solito, per sottrazioni successive, particolarmente semplice da meccanizzare tramite automatismi informatici.

1.7 Casting

Quando si considera un numero binario, risulta fondamentale capire se il valore risulta essere unsigned oppure signed; la differenza è fondamentale perché i due valori

$$\begin{array}{|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 1 & 1 \\ \hline \end{array}$$

in una notazione unsigned differirebbero solamente di Δ minimo, mentre se si trattasse di uan notazione signed, essi sarebbero agli opposti della scala numerica: il primo rappresenta il massimo numero negativo, mentre il secondo è il massimo numero positivo.
Appurata la notazione prescelta, si distinguono le seguenti casistiche

- Nel caso di notazione **unsigned**
  - Per aumentare il numero di cifre decimali, si aggiungono in coda degli 0
  - Per aumentare il numero di cifre intere, si aggiungono in testa degli 0
  - Per ridurre il numero di cifre decimale, si considera il primo bit oltre il range prescelto, e lo si somma all'ultimo bit del range prescelto.
  - Per ridurre il numero di cifre intere, se essi sono 0 non si altera la rappresentazione del numero. Se essi sono 1, si possono scegliere due opzioni: si segnala l'allarme di overflow, oppure si rappresenta il massimo valore possibile con il range di bit a disposizione.
- Nel caso di notazione **signed**:
  - Se il numero è **positivo**
    - Per aumentare il numero di cifre decimali, si aggiungono in coda degli 0
    - Per aumentare il numero di cifre intere, si aggiungono in testa degli 0 (si replica il bit di segno)
    - Per ridurre il numero di cifre decimale, si considera il primo bit oltre il range prescelto, e lo si somma all'ultimo bit del range prescelto.
    - Per ridurre il numero di cifre intere, se essi sono 0 non si altera la rappresentazione del numero. Se essi sono 1, si possono scegliere due opzioni: si segnala l'allarme di overflow, oppure si rappresenta il massimo valore possibile con il range di bit a disposizione (saturazione).
  - Se il numero è **negativo**:
    - Per aumentare il numero di cifre decimali, si aggiungono in coda degli 0
    - Per aumentare il numero di cifre intere, si aggiungono in testa degli 1 (si replica il bit di segno)
    - Per ridurre il numero di cifre decimale, si considera il primo bit oltre il range prescelto, e lo si somma all'ultimo bit del range prescelto.
    - Per ridurre il numero di cifre intere, se essi sono 1 non si altera la rappresentazione del numero. Se essi sono 0, si possono scegliere due opzioni: si segnala l'allarme di overflow, oppure si rappresenta il massimo valore possibile con il range di bit a disposizione (saturazione).

Pertanto, nel caso di notazione **signed**, si replicano i bit di segno per aumentare i bit di rappresentazione, si eliminano senza problemi i bit in testa se essi coincidono con il bit di segno. Per quanto riguarda la parte decimale, non c'è differenza: per aumentare il range di rappresentazione si aggiungono 0, per l'arrotondamento si considera il primo bit oltre il range prescelto, e lo si somma all'ultimo bit del range prescelto.

Osservazione: Nel caso di notazione **signed**, il più piccolo valore positivo è 00000.0001, mentre il più piccolo valore negativo è 11111.1111: in un oscilloscopio, quindi, anche il più piccolo rumore fa saltare l'onda da 00000.0001 a 11111.1111.

1.8 Codici

Un codice è un **insieme di parole** $\mathcal{C}$ adottato per rappresentare gli elementi di un insieme $\mathcal{C}^*$.
I **simboli** sono gli elementi costituenti le parole di codice, mentre la **codifica** è la procedura di associazione di una parola di $\mathcal{C}$ a un elemento di $\mathcal{C}^*$. A tal proposito, si distinguono:

- **Codice non ambiguo**, in cui la corrispondenza tra una parola di $\mathcal{C}$ e un elemento di $\mathcal{C}^*$ è **univoca**;
- **Codice ambiguo**, in cui almeno una parola di $\mathcal{C}$ rappresenta 2 o più elementi di $\mathcal{C}^*$.

Osservazione: Si osservi che se vi sono k simboli, n elementi e le parole sono di lunghezza l, allora il numero di combinazioni possibili è k^l: appare evidente che per non vaere ambiguità deve essere che

$$N \leq k^l \quad \rightarrow \quad \log_k(N) \leq l$$

Pertanto, se

- se $l = \log_k(N)$, allora il codice si dice **efficiente**;
- se $l > \log_k(N)$, allora il codice si dice **ridondante**;
- se $l < \log_k(N)$, allora il codice si dice **ambiguo**;