Logica e Reti Logiche
(Episodio 18: Codifica di numeri frazionari e caratteri)

Francesco Pasquale

18 dicembre 2023

Nell'Episodio 10 abbiamo visto come possiamo rappresentare con sequenze di bit sia numeri positivi che numeri negativi utilizzando la codifica in complemento a due, che è molto comoda per far eseguire somme e sottrazioni a un circuito.

Come possiamo rappresentare invece i numeri frazionari con sequenze di bit? e i caratteri alfabetici e gli altri simboli?

DRAFT

1 Un primo tentativo

Sappiamo che le sequenze distinte di n bit sono 2^n, quindi ognuna di queste potrà rappresentare uno di 2^n "oggetti" diversi. Abbiamo visto che se gli "oggetti" in questione sono i numeri interi senza segno, con la codifica binaria standard usiamo quelle sequenze di bit per indicare i numeri da 0 a $2^n - 1$. Se gli oggetti che vogliamo rappresentare sono i numeri interi positivi e negativi, allora con la *codifica in complemento a due* a n bit indichiamo i numeri interi da -2^{n-1} a $2^{n-1} - 1$.

Per codificare i numeri frazionari, una prima idea potrebbe essere quella di usare una parte dei bit per la parte intera e una parte dei bit per la parte frazionaria. Per esempio, se abbiamo $n = 8$ bit e ne usiamo 4 per la parte intera e 4 per la parte frazionaria, allora la sequenza di bit `01101100` rappresenterebbe il numero

$$\mathtt{0} \cdot 2^3 + \mathtt{1} \cdot 2^2 + \mathtt{1} \cdot 2^1 + \mathtt{0} \cdot 2^0 + \mathtt{1} \cdot 2^{-1} + \mathtt{1} \cdot 2^{-2} + \mathtt{0} \cdot 2^{-3} + \mathtt{0} \cdot 2^{-4} \;=\; (6.75)_{10} \qquad (1)$$

Questo tipo di codifica dei numeri frazionari si chiama *a virgola fissa*.

Esercizio 1. Quando abbiamo definito la codifica binaria dei numeri interi abbiamo visto un metodo che ci consente di scrivere in binario un numero espresso in decimale (si prendono i resti della divisione per *due*, letti in ordine inverso).

Trovare un metodo analogo per codificare in binario un numero compreso fra *zero* e *uno*.

Si può anche usare la codifica a virgola fissa in complemento a due per codificare numeri frazionari positivi e negativi.

Esercizio 2. Quali sono gli intervalli di numeri rappresentabili dai seguenti sistemi numerici?[1]

[1] Potrebbe essere utile ricordare che, se p è un numero reale diverso da 1, allora per ogni $n \geqslant 1$ si ha che $\sum_{i=1}^{n} p^i = \frac{p(1-p^n)}{1-p}$. Se non lo ricordate... dimostratelo per induzione.

1. Numeri in virgola fissa a 32 bit, con 16 bit di parte intera e 16 bit di parte frazionaria.
2. Numeri in complemento a due a 32, bit con 16 bit di parte intera e 16 bit di parte frazionaria.

L'utilizzo della codifica in virgola fissa non è in genere molto conveniente, perché non ci permette di lavorare con numeri molto grandi e numeri molto piccoli.

Un modo più saggio di usare i bit che abbiamo a disposizione viene dall'utilizzo della *notazione scientifica*: osservate che, se dobbiamo scrivere in decimale numeri come *novecentottantasette miliardi* oppure *sessantaquattro miliardesimi*, tipicamente non li scriveremo così:

$$987000000000 \qquad \text{e} \qquad 0.000000064$$

ma così:

$$9.87 \cdot 10^{11} \qquad \text{e} \qquad 6.4 \cdot 10^{-8}$$

Chiaramente ci sono più modi di scrivere lo stesso numero in notazione scientifica (per esempio, avrei potuto scrivere il primo numero anche come $98,7 \cdot 10^{10}$). Quando si usa un'unica cifra prima della virgola, si dice che la notazione scientifica è *normalizzata*.

Anche in binario possiamo fare la stessa cosa. Per esempio, possiamo scrivere il numero in (1) in notazione scientifica normalizzata così

$$1.1011 \cdot 2^2$$

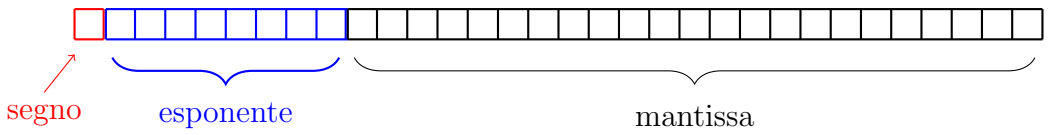
Esercizio 3. Osservate che quando scriviamo un numero in binario in notazione scientifica normalizzata il numero prima della virgola sarà sempre 1.

Se abbiamo a disposizione n bit quindi, invece che usarne alcuni per la parte intera e alcuni per quella frazionaria, possiamo decidere di usarne alcuni per l'*esponente* e alcuni per la *mantissa*.

Nella prossima sezione vediamo qual è lo standard attuale che definisce come utilizzare i bit che abbiamo a disposizione.[2].

2 Lo standard *IEEE754*

Lo standard IEEE-754 per i numeri in virgola mobile a *precisione singola* prevede 32 bit:
- Il primo bit per il segno (0 positivo, 1 negativo);
- I successivi 8 bit per l'esponente, espresso in codifica ad eccesso (ossia, si aggiunge 127 al numero decimale da codificare in binario);
- I successivi 23 bit per la mantissa, di cui non si memorizza l'uno più significativo.



Per esempio, supponiamo di dover codificare $-76,28125$. È un numero negativo, quindi il primo bit sarà 1. La parte intera è $76 = 64 + 8 + 4 = 2^6 + 2^3 + 2^2$ quindi,

[2]Per esempio, il tipo di dati *float* in C segue questo standard

DRAFT

in binario, 1001100. La parte frazionaria è $0,28125 = 2^{-2} + 2^{-5}$ quindi, in binario 0.01001. Il valore assoluto del nostro numero in binario perciò è: $1001100.01001 = 1.00110001001 \times 2^6$. Perciò la mantissa escluso l'uno più significativo è `00110001001`, a cui andremo ad aggiungere tutti gli zeri che servono per arrivare a 23 bit. L'esponente è 6, che in codifica ad eccesso diventa $6 + 127 = 133$, ossia `10000101` in binario a otto bit. Complessivamente quindi la codifica del nostro numero $-76,28125$ sarà

`11000010100110001001000000000000`

che espresso in esadecimale diventa `1100 0010 1001 1000 1001 0000 0000 0000` = `C2989000`.

Esercizio 4. Scrivere i numeri seguenti in virgola mobile secondo lo standard IEEE 754 a precisione singola. Scrivere il risultato in esadecimale

(a) -13.5625 *(b)* 42.3125 *(c)* -17.15625.

Esercizio 5. Scrivere in decimale i seguenti numeri in virgola mobile in formato IEEE 754 a precisione singola espressi in esadecimale

(a) `C0123000` *(b)* `81C56000` *(c)* `D0B10301`.

Esercizio 6. Osservate che codificare lo *zero* in questo modo sarebbe impossibile.

Alcune sequenze di 32 bit sono riservate a codificare numeri speciali:

- `X 00000000 00000000000000000000000` rappresenta lo *zero* (dove X può essere 0 o 1)
- `0 11111111 00000000000000000000000` rappresenta $+\infty$
- `1 11111111 00000000000000000000000` rappresenta $-\infty$
- Una sequenza in cui i bit dell'esponente sono tutti 1 ma quelli della mantissa non sono tutti 0 non rappresenta nessun numero (*NaN - Not a Number*)

Lo standard IEEE754 definisce anche una codifica a 64 bit per i numeri in virgola mobile.

Esercizio 7. Per numeri in virgola mobile a *precisione doppia* vengono usati 64 bit: uno per il segno, 11 per l'esponente e i restanti per la mantissa. Qual è il numero da sommare all'esponente per ottenere la codifica ad eccesso?

DRAFT

3 Cenni alla codifica dei caratteri: *ASCII*, *Unicode*, UTF-8

Oltre ai numeri interi e frazionari, abbiamo bisogno di codificare in binario anche tutti gli altri caratteri. Abbiamo già incontrato in una precedente esercitazione il codice *ASCII (American Standard Code for Information Interchange)* che, sviluppato negli anni sessanta, è stato il primo standard a imporsi a livello internazionale. La codifica *ASCII* usa 7 bit (un *byte* con il primo bit a 0) per codificare 128 caratteri.

La necessità di utilizzare ben più di 128 caratteri ha portato, negli anni, alla definizione di diversi standard nelle diverse parti del mondo. Per superare le difficoltà di interazione fra sistemi che utilizzano codifiche diverse, verso la fine degli anni ′80 ha iniziato a prendere forma quello che oggi è uno standard unificante a livello internazionale: *Unicode*. Unicode associa ad ogni carattere un numero compreso fra 0 e 1114111 (fra `000000` e `10FFFF`, in esadecimale) per un totale di $2^{16} + 2^{20}$ caratteri utilizzabili, e può essere implementato con diverse codifiche (ossia non c'è un unico modo di associare il numero che identifica il carattere a una sequenza di bit).

Le codifiche più utilizzate per implementare Unicode sono le UTF *(Unicode Transformation Format)*: UTF-8, UTF-16, UTF-32. La UTF-32 è una codifica a *lunghezza fissa* che usa 32 bit per rappresentare in binario ognuno dei numeri dell'inervallo utilizzato da *Unicode*. Quella più diffusa tuttavia è la codifica UTF-8, che è una codifica *a lunghezza variabile* (alcuni caratteri vengono codificati con 8 bit, altri con 16, altri con 24, altri con 32). I 128 caratteri del codice *ASCII* vengono codificati in UTF-8 con 8 bit (un byte): il primo bit a 0 e gli stessi 7 bit utilizzati dal codice *ASCII*

0 - - - - - - -

Quindi UTF-8 è *retrocompatibile* con *ASCII*: ogni sequenza di bit che si può decodificare con *ASCII*, si può decodificare allo stesso modo anche con UTF-8. Per i caratteri codificati con due, tre o quattro *byte*, il primo byte comincia con una sequenza di 1 lunga quanto il numero di *byte* utilizzati per codificare il carattere, gli altri *byte* cominciano con 10, i bit che restano sono utilizzati per la codifica binaria del numero che Unicode associa al carattere da codificare

2 byte	1 1 0 - - - - -	1 0 - - - - - -		
3 byte	1 1 1 0 - - - -	1 0 - - - - - -	1 0 - - - - - -	
4 byte	1 1 1 1 0 - - -	1 0 - - - - - -	1 0 - - - - - -	1 0 - - - - - -

DRAFT