

Sistema numerico binario

Il **sistema numerico binario** è un sistema numerico posizionale in base 2. Esso utilizza solo due simboli, di solito indicati con 0 e 1, invece delle dieci cifre utilizzate dal sistema numerico decimale. Ciascuno dei numeri espressi nel sistema numerico binario è chiamato "numero binario".

In informatica il sistema binario è utilizzato per la rappresentazione interna dell'informazione dalla quasi totalità degli elaboratori elettronici, in quanto le caratteristiche fisiche dei circuiti digitali rendono particolarmente conveniente la gestione di due soli valori.

Rappresentazione

Un numero binario è una sequenza di cifre binarie (dette bit). Ogni cifra in posizione n (contate da destra verso sinistra iniziando da 0) si considera moltiplicata per 2^n , anziché per 10^n , come avviene nella numerazione decimale.

Binario	Decimale
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10

La formula per convertire un numero da binario a decimale (dove con d_n si indica la cifra di posizione n all'interno del numero, contate da destra verso sinistra iniziando da 0) è

$$d_n 2^n + d_{n-1} 2^{n-1} + \dots + d_1 2^1 + d_0 2^0 = N_{10}$$

Ad esempio

$$1001_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 9_{10} .$$

L'utilizzo dei numeri binari non è ristretto esclusivamente alla rappresentazione dei numeri interi positivi. Adottando alcune convenzioni, è possibile rappresentazione dei numeri interi relativi in binario.

Complemento a due

Il **complemento a due** (in inglese *two's complement*) è il metodo più diffuso per la rappresentazione dei numeri con segno in informatica. Col complemento a due, il bit iniziale (più a sinistra) del numero ha peso negativo o positivo; da questo deriva che tutti i numeri che cominciano con un "1" sono numeri binari negativi, mentre tutti i numeri che cominciano con uno "0" sono numeri binari positivi. Si può così ottenere il valore assoluto di un numero binario negativo, prendendo il complementare (invertendo il valore dei singoli bit) e aggiungendo 1 al numero binario risultante.

Un numero binario di n cifre può rappresentare con questo metodo i numeri compresi fra -2^{n-1} e $+2^{n-1}-1$, così un binario di 8 cifre può rappresentare i numeri compresi tra -128 e +127.

Questo metodo consente di avere un'unica rappresentazione dello zero e di operare efficientemente addizione e sottrazione sempre avendo il primo bit a indicare il segno.

Caratteristiche

Questo metodo di rappresentazione ha notevoli vantaggi, soprattutto per effettuare somme e differenze: in pratica ai numeri viene anteposto un bit di valore zero; se poi il numero è negativo è necessario convertirlo in complemento a 2: per farlo è sufficiente leggere il numero da destra verso sinistra e invertire tutte le cifre a partire dal primo bit uguale a 1 (escluso), trasformando gli 0 in 1 e viceversa. Per fare un esempio:

Come è possibile notare seguendo questo metodo il primo bit diventa automaticamente il bit del segno (come per la rappresentazione a modulo e segno). Viene però risolto il problema dell'ambiguità dello 0 (in complemento a 2 00000 e 10000 hanno significati diversi) e vengono enormemente facilitate le operazioni di somma e differenza, che si riducono alla sola operazione di somma: per spiegare meglio basta fare un esempio:

Calcolo dell'opposto in complemento a due

Per rappresentare l'opposto di un numero binario in complemento se ne invertono, o negano, i singoli bit: si applica cioè l'operazione logica NOT. Si aggiunge infine 1 al valore del numero trovato con questa operazione.

Facciamo un esempio rappresentando il numero -5 con 8 bit in complemento a 2.

Partiamo dalla rappresentazione in binario del numero 5:

```
0000 0101 (5)
```

La prima cifra è 0, quindi il numero è sicuramente positivo. Invertiamo i bit: 0 diventa 1, e 1 diventa 0:

```
1111 1010
```

A questo punto abbiamo ottenuto il complemento a uno del numero 5; per ottenere il complemento a due sommiamo 1 a questo numero:

```
1111 1010 + 0000 0001 = 1111 1011 (-5)
```

Il risultato è un numero binario con segno che rappresenta il numero negativo -5 secondo il complemento a due. Il primo bit, pari a 1, evidenzia che il numero è negativo.

Il complemento a due di un numero negativo ne restituisce il numero positivo pari al valore assoluto: invertendo i bit della rappresentazione del numero -5 (sopra) otteniamo:

```
0000 0100
```

Sommando 1 otteniamo:

```
0000 0100 + 0000 0001 = 0000 0101 (+5)
```

Che è appunto la rappresentazione del numero +5 in forma binaria.

Si noti che il complemento a due dello zero è zero stesso: invertendone la rappresentazione si ottiene un byte di 8 bit pari a 1, e aggiungendo 1 si ritorna a tutti 0 (l'overflow viene ignorato).

Addizione

Operare l'addizione di due interi rappresentati con questo metodo non richiede processi speciali se essi sono di segno opposto, e il segno viene determinato automaticamente. Facciamo un esempio addizionando 15 e -5:

```
11111 1110 (riporto)
 0000 1111 (15)
+ 1111 1011 (-5)
=====
 0000 1010 (10)
```

Questo processo gioca sulla lunghezza fissa di 8 bit della rappresentazione: viene ignorato un riporto di 1 che causerebbe un overflow, e rimane il risultato corretto dell'operazione (10).

Gli ultimi due bit (da destra a sinistra), ovvero i più significativi, della riga dei riporti contengono importanti informazioni sulla validità dell'operazione: se il risultato è compreso o non è compreso nell'intervallo dei numeri rappresentabili. Si verifica se il riporto è stato eseguito sul bit del segno ma non è stato portato fuori, o viceversa; più semplicemente, se i due bit più a sinistra sulla riga dei riporti non sono entrambi 0 o 1. Per verificare la validità del risultato è conveniente eseguire su questi due bit un'operazione XOR. Vediamo un esempio di addizione a 4 bit di 7 e 3:

```
01110 (riporto)
 0111 (7)
```

```

+ 0011  (3)
=====
1010  (-6)

```

In questo caso, come si può notare dal riporto presente solo sul bit più significativo, si è in presenza di *overflow*, per cui il risultato non è 10 (come sarebbe corretto) ma -6, infatti il massimo numero positivo rappresentabile in complemento a due su quattro bit è 7 (con $n=4$: $2^{(n-1)} - 1 = 7$).

Sottrazione

Anche se la sottrazione potrebbe essere eseguita aggiungendo il complemento a due del sottraendo al minuendo, questo procedimento è poco utilizzato in quanto porta più complicazioni che semplicemente costruire un circuito per la sottrazione. Ma come per l'addizione, il vantaggio del complemento a due è l'eliminazione della necessità di esaminare i segni degli operandi per determinare quale operazione sia necessaria. Per esempio, sottrarre -5 a 15 è come aggiungere 5 a 15, ma questo è nascosto dal complemento a due:

```

1111 0000  (riporto)
0000 1111  (15)
- 1111 1011  (-5)
=====
0001 0100  (20)

```

L'overflow viene individuato con lo stesso metodo usato per l'addizione, esaminando i due bit più a sinistra sulla riga dei riporti: se sono differenti si è verificato un overflow.

Facciamo un altro esempio con una sottrazione con risultato negativo: $15 - 35 = -20$:

```

1110 0000  (riporto)
0000 1111  (15)
- 0010 0011  (35)
=====
1110 1100  (-20)

```

Moltiplicazione

Moltiplicazione di numeri senza segno

Il prodotto di due numeri senza segno di n cifre può essere rappresentato con $2n$ cifre, quindi il prodotto di due numeri da 8 bit viene rappresentato con 16 bit. Nel sistema binario la moltiplicazione del moltiplicando per un bit del moltiplicatore è facile.

Se il bit del moltiplicatore è 1, si copia il moltiplicando facendolo scorrere nella posizione appropriata. Se il bit del moltiplicatore è 0, si inseriscono zeri.

Il prodotto è calcolato un bit alla volta sommando le colonne di bit da destra a sinistra propagando i valori del riporto fra le colonne.

Moltiplicazione di numeri con segno

Si discute adesso la moltiplicazione di operandi in complemento a due, che genera un prodotto di lunghezza doppia. La strategia generale è ancora di accumulare i prodotti parziali sommando versioni del moltiplicando selezionate dai bit del moltiplicatore.

Si consideri inizialmente il caso di un moltiplicatore positivo e un moltiplicatore negativo.

Quando si somma un moltiplicando negativo a un prodotto parziale, bisogna estendere il valore del bit di segno del moltiplicando a sinistra per tutta l'estensione del prodotto.

Esempio :

$$\begin{array}{r}
 10011 \text{ } (-13) \\
 \times 01011 \text{ } (+11) \\
 \hline
 1111110011 \\
 111110011 \\
 000000000 \\
 1110011 \\
 0000000 \\
 \hline
 1101110001 \text{ } (-143)
 \end{array}$$

In questo esempio, il moltiplicando è un operando con segno a 5 bit, -13. Esso è moltiplicato per +11, ottenendo il prodotto a 10 bit, -143.

Per un moltiplicatore negativo, la soluzione immediata è formare il complemento a due di entrambi moltiplicatore e moltiplicando e procedere come nel caso di un moltiplicatore positivo. Questo è possibile poiché complementando entrambi gli operandi non cambiamo né valore né segno del prodotto.

Divisione di interi

Analogamente alla moltiplicazione, si segue lo stesso approccio nel discutere la divisione di interi.

Si discute la divisione di numeri senza segno in dettaglio, quindi si inserisce qualche commento generale sul caso dei numeri con segno.

Esempio :

21	1 0 1 0 1
13) ---	
274	1 1 0 1) 1 0 0 0 1 0 0 1 0
26	1 1 0 1
-----	-----
14	1 0 0 0 0
13	1 1 0 1
-----	-----
1	1 1 1 0
	1 1 0 1

	1

L'esempio mostra una divisione decimale e binaria degli stessi valori. Si consideri prima la versione decimale. Il 2 nel quoziente è determinato dal seguente ragionamento: prima, si prova a dividere 2 per 13, il che non è possibile. Quindi, si prova a dividere 27 per 13. Si procede nel tentativo moltiplicando il 13 per 2, che dà 26, e osservando che 27-26=1 è meno di 13, si pone 2 a quoziente e si svolge la sottrazione richiesta. Si abbassa la prossima cifra del dividendo, 4, e si finisce decidendo che 13 sta in 14 una volta con il resto di 1. E' possibile discutere la divisione binaria in modo simile, con la semplificazione che le sole possibilità per i bit del quoziente sono 0 e 1.

L'algoritmo (o circuito) che realizza la divisione con questo metodo manuale opera come segue : posiziona il divisore in modo appropriato rispetto al dividendo e svolge una sottrazione. Se il resto è 0 o positivo, si determina 1 per un bit del quoziente, si estende il resto con un altro bit del dividendo, si riposiziona il divisore e si procede svolgendo un'altra sottrazione. Se il resto è negativo, si determina 0 per il bit del quoziente, si ripristina il dividendo sommandogli il divisore appena sottratto e si riposiziona il divisore per un'altra sottrazione. Questo è detto algoritmo di divisione con ripristino (restoring division).