DENTRO LA SCATOLA

Rubrica a cura di

Fabio A. Schreiber

Il Consiglio Scientifico della rivista ha pensato di attuare un'iniziativa culturalmente utile presentando in ogni numero di Mondo Digitale un argomento fondante per l'Informatica e le sue applicazioni; in tal modo, anche il lettore curioso, ma frettoloso, potrà rendersi conto di che cosa sta "dentro la scatola". È infatti diffusa la sensazione che lo sviluppo formidabile assunto dal settore e di conseguenza il grande numero di persone di diverse estrazioni culturali che - a vario titolo - si occupano dei calcolatori elettronici e del loro mondo, abbiano nascosto dietro una cortina di nebbia i concetti basilari che lo hanno reso possibile. La realizzazione degli articoli è affidata ad autori che uniscono una grande autorevolezza scientifica e professionale a una notevole capacità divulgativa.

Le operazioni aritmetiche

Luigi Ciminiera

1. Introduzione

Questo è il secondo di una serie di articoli, di cui il primo è apparso nello scorso numero [1], che sono dedicati ad alcuni aspetti fondamentali di un sistema di elaborazione, visto come calcolatore, ovvero come macchina per effettuare calcoli. L'articolo precedente si è soffermato sulle rappresentazioni binarie, concentrandosi soprattutto sul modo in cui si rappresentano i numeri. In questo secondo articolo della serie ci si occuperà di descrivere gli algoritmi più comuni per l'effettuazione delle operazioni di somma/sottrazione, moltiplicazione e divisione.

L'algoritmo relativo a una qualsiasi delle operazioni aritmetiche non può prescindere dalla rappresentazione nell'articolo precedente, ovvero:

- **Numeri binari senza segno**, espressi utilizzando *n* bit come:

$$X = \sum_{i=0}^{n-1} x_i 2^i$$

- **Numeri in modulo e segno** che, ricordando che il bit x_{n-1} è quello che esprime il segno negativo (1) o positivo (0), sono espressi come:

$$X = (-1)^{x_{n-1}} \sum_{i=0}^{n-2} x_i 2^i$$

- **Numeri in complemento a 2**, espressi come:

$$X = - x_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

Questo articolo si limita, inoltre, alla trattazione delle operazioni per i numeri a virgola fissa, rimandando alla letteratura citata in bibliografia, per le operazioni con i numeri a virgola mobile, molto diverse da quelle qui discusse, soprattutto a causa del numero molto maggiore di gradi di libertà.

2. Somma e sottrazione

L'operazione di somma con i numeri interi positivi deve essere effettuata in binario nello stesso modo in cui viene effettuata in decimale: si parte dalla colonna a destra e si calcola, per ogni colonna, la somma delle cifre corrispondenti degli addendi, più un eventuale riporto generato dalla precedente colonna. Per realizzare questo semplice algoritmo, è necessario però conoscere il valore della cifra di somma e quello del riporto che vengono generate da tutte le possibili combinazioni delle cifre nella base prescelta, che per la base 2, sono i seguenti:

0 + 0 + 0 = 0 *con riporto di* 0
1 + 0 + 0 = 0 + 1 + 0 = 0 + 0 + 1 = 1 *con riporto di* 0
1 + 1 + 0 = 1 + 0 + 1 = 0 + 1 + 1 = *o con riporto di* 1
1 + 1 + 1 = 1 *con riporto di* 1

Gli esempi seguenti mostrano come viene effettuata la somma dei numeri decimali 3 + 7 e 10 + 8, utilizzando una rappresentazione su 4 bit.

riporti	1110
3_{10}	0011
7_{10}	0111
10_{10}	1010

riporti	10000
10_{10}	1010
8_{10}	1000
2_{10}	0010

Nell'esempio di destra, si può notare come la somma produca un riporto generato nella colonna dei bit più significativi, che deborda a sinistra dalla nostra rappresentazione di 4 bit. Questo riporto in uscita a sinistra è l'indicazione che si è prodotto un *overflow*, ovvero il risultato è talmente grande che non può più essere rappresentato con il numero di bit prescelto; d'altra parte, è facile verificare che 4 bit consentono, in questa rappresentazione, di operare con numeri interi da 0 a 15, mentre il risultato corretto è 18, che richiederebbe almeno 5 bit.
Per poter essere sicuri di non incorrere in overflow, bisogna garantire che la rappresentazione della somma abbia almeno 1 bit in più rispetto a quella dell'operando più lungo. Nel caso di somma di k numeri da n bit, la lunghezza minima della rappresentazione del risultato, che garantisca contro gli overflow è $n + \lceil \log_2 k \rceil$.
Per quanto riguarda la sottrazione, le regole sono leggermente differenti: si procede sempre da destra verso sinistra, a ogni colonna bisogna sottrarre al bit del primo operando quello del secondo operando e anche un eventuale prestito utilizzato nella colonna precedente.

0 – 0 – 0 = 1 – 1 – 0 = 1 – 0 – 1 = 0 *con prestito di* 0
0 – 1 – 0 = 0 – 0 – 1 = 1 – 1 – 1 = 1 *con prestito di* 1
1 – 0 – 0 = 1 *con prestito di* 0
0 – 1 – 1 = 0 *con prestito di* 1

Gli esempi che seguono mostrano come vengono effettuate le sottrazioni 9 – 7 e 5 – 7, utilizzando le regole precedenti.

prestiti	01000
9_{10}	1001
5_{10}	0101
4_{10}	0100

prestiti	11100
5_{10}	0101
7_{10}	0111
14_{10}	1110

Anche in questo caso, l'esempio di destra mostra un'anomalia segnalata dal fatto che vi è un prestito in uscita dalla colonna più a sinistra. In questo caso, il problema consiste nel fatto che il risultato è negativo (–2), mentre si sta utilizzando una rappresentazione che considera solo, per definizione, numeri interi *positivi*. Per poter trattare anche i numeri interi negativi, bisogna ricorrere a una delle altre due rappresentazioni considerate. In particolare, la rappresentazione in complemento a 2 permette di effettuare le somme utilizzando esattamente lo stesso algoritmo precedentemente illustrato per i numeri senza segno, con l'ulteriore semplificazione che anche i bit di segno vengono trattati nello stesso modo rispetto a tutti gli altri bit. Si può, quindi, procedere come mostrano gli esempi riportati di seguito:

riporti	11110
$+3_{10}$	0011
-1_{10}	1111
$+2_{10}$	0010

riporti	10000
-6_{10}	1010
-8_{10}	1000
$+2_{10}$	0010

Come sempre, l'esempio a destra mostra un'anomalia, che in questo caso non è dovuta al riporto uscente dalla posizione più a sinistra (quella dei bit di segno): infatti, l'esempio di sinistra mostra anch'esso un riporto uscente a sinistra, ma si ottiene il risultato corretto ignorando semplicemente questo riporto. Il vero problema, messo in rilievo dall'esempio di destra, consiste nel fatto che sommando 2 operandi negativi si è ottenuto un risultato positivo. In complemento a 2, questa è la prova che è avvenuto un overflow; andando a controllare si può verificare che il risultato corretto è –14, mentre la rappresentazione su 4 bit utilizzata permette di manipolare solo i numeri interi fra –8 e +7. Analogamente, si ha overflow anche se sommando 2 numeri positivi si ottiene un risultato negativo. Si noti che, nelle rappresentazioni di numeri con segno, non è necessario trattare il caso della sottrazione, dal momento che questa viene effettuata sommando l'opposto del secondo operando, anziché procedere con la sottrazione diretta.
Una possibile alternativa al complemento a 2 è data dalla rappresentazione in modulo e segno. Bisogna, innanzitutto, riconoscere che questo tipo di rappresentazione è esattamente la stessa (a parte il cambio di base) che utilizziamo noi umani per la base 10. In binario, tutte le cifre hanno 2 valori come pure il segno (+ o –), per cui è venuto naturale rappresentare gli stessi segni mediante dei bit (0 per +, 1 per –);

invece, nel caso dei numeri decimali, le cifre hanno 10 valori e il segno solo 2 per cui sono stati adottati simboli diversi.
Anche le regole per la somma sono quelle che si adottano normalmente nei calcoli manuali: se i segni degli operandi sono concordi, bisogna sommare i moduli e il segno del risultato è quello degli operandi, se i segni sono discordi, bisogna sottrarre il modulo inferiore da quello superiore e il segno è determinato dall'addendo con modulo superiore.
L'automazione di queste regole fa sorgere, però, un problema: determinare l'operando con modulo maggiore comporta già di per sé una sottrazione. Per semplificare l'algoritmo nel caso di segni discordi, si adottano le seguenti regole: si sottrae il modulo del secondo operando dal primo; se il prestito uscente a sinistra è 0, il primo operando aveva effettivamente modulo maggiore, altrimenti è il contrario, e il modulo corretto del risultato viene ottenuto complementando a 2 il risultato ottenuto (quest'ultima operazione si effettua invertendo tutti i bit e sommando al risultato 1 nella posizione più a destra).
Situazioni di overflow possono presentarsi solo nel caso di somma dei moduli (operandi con segni concordi) e vengono rivelate dal fatto che questa somma produce un riporto uscente a sinistra. Per quanto riguarda la lunghezza minima della rappresentazione del risultato che garantisce contro possibili overflow, si applicano le stesse formule valide per i numeri senza segno.

3. Moltiplicazione

Anche per la moltiplicazione è possibile partire da una rivisitazione dell'algoritmo adottato nelle operazioni manuali con numeri decimali, iniziando a considerare il caso dei numeri senza segno. Il meccanismo di moltiplicazione consiste nel partire dalla cifra meno significativa del moltiplicatore, moltiplicare tutto il moltiplicando per quella cifra, ottenendo un primo risultato parziale; si passa a moltiplicare tutto il moltiplicando per la successiva cifra del moltiplicatore, il risultato viene sommato a quello parziale spostandolo di una posizione a sinistra. Si continua così a moltiplicare ogni cifra del moltiplicatore per il moltiplicando, e a sommare il risultato spostandosi ogni volta di una posizione a sinistra.
Per poter effettuare la moltiplicazione di una cifra del moltiplicatore per il moltiplicando, è necessario servirsi della famosa *tavola pitagorica*, che dice qual è il risultato della moltiplicazione per ciascuna delle possibili coppie di cifre della rappresentazione. Nel caso della base 2, la tavola pitagorica si riduce alla seguente forma:

	0	1
0	0	0
1	0	1

L'overflow è possibile anche nella moltiplicazione. Per il caso dei numeri senza segno, la lunghezza minima del prodotto, che garantisce di poter rappresentare qualsiasi risultato, è di $m + n$ bit, dove m e n sono le lunghezze in bit delle rappresentazioni dei due fattori.
La moltiplicazione di numeri in modulo e segno è sostanzialmente identica a quella dei numeri senza segno, poiché il modulo del prodotto è il prodotto dei moduli (che sono numeri positivi senza segno), mentre il segno del risultato viene ottenuto dal semplice EXOR dei due bit di segno dei fattori. L'unica variazione da notare è che la moltiplicazione di un fattore lungo m bit per uno di lunghezza n bit, produce un prodotto che occupa al massimo $m + n - 1$ bit.
Per i numeri in complemento a 2, bisogna rifarsi alla formula data all'inizio di questo articolo. La moltiplicazione può avvenire come quella dei numeri senza segno, tenendo però presente la seguente particolarità della rappresentazione. Il bit più significativo del moltiplicatore ha un peso negativo (-2^{n-1}), per cui, se questo bit è a 1, bisogna sottrarre il moltiplicando (o sommare il suo complemento a 2), anziché sommarlo.
Nel caso di fattori di lunghezza m e n bit rispettivamente, il prodotto occuperà $n + m$ bit al massimo. È bene notare che non si può usare 1 bit in meno, per via di un'unica combinazione nei valori dei fattori che richiede il massimo di bit; questo unico caso corrisponde alla moltiplicazione $(-2^{n-1}) \times (-2^{m-1}) = +2^{m+n-2}$, che richiede, appunto, una rappresentazione di almeno $n + m$ bit.
L'esempio seguente mostra la moltiplicazione $(-5_{10}) \times (-3_{10})$, con fattori su 4 bit, che in complemento a 2 sono rappresentati come $X = 1011$ (moltiplicando) e $Y = 1101$ (moltiplicatore). Poiché il moltiplicando è negativo, bisognerà som-

mare una serie di addendi negativi con una rappresentazione su 8 bit, che è la massima necessaria per il prodotto dei due operandi da 4 bit. In complemento a 2, l'allungamento della rappresentazione si effettua ripetendo a sinistra il segno del numero (cioè il bit più a sinistra), per questo motivo i vari addendi negativi appaiono nell'esempio seguente in una rappresentazione allungata rispetto a quella originale del moltiplicando.

	1011 x
	1101
	11111011
	0000000
	111011
Sottrazione del moltiplicando W	00101
Risultato= 15_{10}	00001111

4. Divisione

Contrariamente alle altre operazioni aritmetiche viste in precedenza, non è sempre possibile calcolare esattamente il quoziente di una divisione. Il motivo risiede nel fatto che la divisione, anche se ci si limita a operandi interi, non produce necessariamente un risultato che ha una rappresentazione con numero finito di cifre.
In questo contesto, verrà esaminata la divisione intera, in quanto l'algoritmo per il caso generale è lo stesso, ma varia solo il momento in cui si decide di interrompere il calcolo dei bit del quoziente. Nella divisione intera, il dividendo *X*, il divisore *D*, il quoziente *Q* e il resto *W* sono legati dalla seguente relazione, dove tutte le quantità sono intere:

$$X = QD + W \qquad 0 \le |W| < |D|$$

Un ulteriore vincolo è rappresentato dal fatto che il resto abbia lo stesso segno del dividendo. Anche in questo caso, l'algoritmo utilizzato potrebbe essere quello ben noto della base 10, con gli opportuni adattamenti alla base 2. Questo algoritmo viene detto di tipo *restoring*, in quanto bisogna ripristinare il valore del resto parziale, qualora questo assuma valore negativo a seguito di una delle sottrazioni che vengono effettuate.
La velocità di esecuzione può essere migliorata se si permette di ottenere un risultato negativo dopo qualsiasi sottrazione; ovviamente, al passo successivo bisognerà tenere conto di questo fatto. Il nuovo algoritmo che si ottiene è detto di tipo *non-restoring*, in quanto non si disfa mai l'operazione di sottrazione/somma appena fatta. La divisione non-restoring per numeri positivi può essere descritta nel modo seguente, dove W_i è l'*i*-esimo valore assunto dal resto parziale e q_i è l'*i*-esimo bit del quoziente; si noti, inoltre, che vale l'ipotesi che il dividendo sia rappresentato su 2*n* bit e il divisore su *n* bit.

$W_0 = X$
$W_1 = 2\,W_0 - D\,2^n$
for $j = 1 \ldots . n-1$
  if $W_j \ge 0$ **then** $q_{n-j} = 1$; $W_{j+1} = 2\,W_j - D\,2^n$;
    else $q_{n-j} = 0$; $W_{j+1} = 2\,W_j + D\,2^n$;
endfor
if $W_n \ge 0$ **then** $q_0 = 1$;
  else $q_0 = 0$; $W_n = W_n + D\,2^n$;
    [correzione del segno del resto]

Nell'algoritmo sopra descritto il resto parziale viene scalato di una posizione a sinistra (moltiplicazione per 2) a ogni passo, anziché far scalare di una posizione a destra la quantità da sottrarre, come avviene quando la divisione viene effettuata manualmente.
Il numero di bit interi del quoziente può essere derivato dalla sottrazione del numero dei bit della rappresentazione del dividendo meno quelli della rappresentazione del divisore, come avviene anche in decimale; il numero ottenuto rappresenta un valore massimo. Nell'esempio seguente, viene mostrato il funzionamento dell'algoritmo non-restoring per la divisione di 11_{10} = 00001011_2 (dividendo) per 2_{10} = 0010_2 (divisore); si noti che viene effettuata una divisione di un numero da 8 bit per un altro da 4 bit, di conseguenza la parte intera del quoziente non richiederà più di 4 bit.
Nella colonna più a sinistra viene anche mostrato il valore del riporto uscente a sinistra, che indica il segno del risultato delle varie operazioni, si noti anche che le sottrazioni vengono effettuate sommando il complemento a 2 del divisore.

$W_0 =$ 0 0000 1011

$2\,W_0 =$ 0 0001 0110

$-D\,2^4 =$ 1 1110

$W_1 =$ 1 1111 0110 $q_3 = 0$

$2\,W_1 =$ 1 1110 1100

$+D\,2^4 =$ 0 0010

$W_2 =$ 0 0000 1100 $q_2 = 1$

$2\,W_2 =$ 0 0001 1000

$-D\,2^4 =$ 1 1110

$W_3 =$ 1 1111 1000 $q_1 = 0$

$2\,W_3 =$ 1 1111 0000

$+D\,2^4 =$ 0 0010

$W_4 =$ 0 0001 0000 $q_0 = 1$

resto = 0 0001

Il quoziente è dato da $0101_2 = 5_{10}$ mentre il resto finisce nella parte superiore di *W* ed è pari a 1.

Bibliografia

[1] Dadda L. : Fondamenti dell'aritmetica digitale: i codici numerici. *Mondo Digitale*, Anno III, n. 9, marzo 2004, p. 61-65.

[2] Ercegovac M.D., Lang T.: *Digital Arithmetic*. Morgan Kaufmann, 2004.

[3] Koren I.: *Computer Arithmetic Algorithms*. Prentice-Hall, 1993.

Luigi Ciminiera si è laureato in Ingegneria Elettronica nel 1977 presso il Politecnico di Torino. Dal 1979 ha iniziato a collaborare alle attività del Dipartimento di Automatica e Informatica, presso il quale ha ricoperto varie posizioni. Attualmente è professore ordinario di Sistemi di Elaborazione dell'Informazione. Dal 1989, è membro del Comitato di Programma di IEEE Symposium on Computer Arithmetic, di cui è anche stato Program Co-Chairman nell'edizione del 2001. Oltre all'aritmetica per elaboratori, i suoi interessi scientifici comprendono anche i sistemi peer-to-peer e le griglie computazionali. È stato co-autore di 2 libri a diffusione internazionale e di otre 100 contributi pubblicati in riviste e conferenze scientifiche. Egli ricopre attualmente la carica di Preside della II Facoltà di Ingeneria del Politecnico di Torino.
luigi.ciminiera@polito.it

ERRATA CORRIGE

Sul numero 7 di marzo 2003 di Mondo Digitale, nell'articolo "Aspettando Robot" sono state pubblicate, a pagina 15, due immagini con didascalie errate. Riportiamo, qui sotto le due didascalie corrette e complete di fonte.

Figura 9 (sopra)
La cella robotica sottomarina realizzata dal CNR-IAN-Reparto Robotica, Unige-Dist e Ansaldo, nell'ambito del Progetto Europeo 1996/1998 EC MAS3 CT950024 AMADEUS (Advanced Manipulation for DEep Underwater Sampling) (foto G. Veruggio).

Figura 9 (sotto)
Il robot sottomarino Romeo realizzato dal Reparto Robotica del CNR-IAN, durante le prove finali del Progetto Europeo 1997/2000 EC MAS3 CT970083 ARAMIS (Advanced Rov package for Automatic Mobile Investigation of Sediments) (foto G. Veruggio).