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 "macchine" aritmetiche

Bruno Fadini Roberto Canonico

1. Introduzione

Così come le macchine meccaniche trasformano energia, le "macchine informatiche" trasformano informazioni: esse forniscono uno (o più) dati in output che sono la trasformazione, secondo regole assegnate, dei dati in input. Una macchina aritmetica è, coerentemente, una le cui regole sono quelle dell'aritmetica: addizionatori, sottrattori, moltiplicatori ecc. sono le "macchine aritmetiche".

Di una macchina aritmetica si può definire "cosa fa" (per esempio, un addizionatore calcola $z = x + y$) oppure "come lo fa" (ovvero, calcola la somma bit per bit) oppure ancora "come è fatta" (per esempio, mediante blocchi logici o elementi circuitali): si tratta di diversi "livelli di astrazione" della macchina, il primo ne definisce gli aspetti concettuali, l'ultimo quelli costruttivi, spesso legati a problemi logico-elettronici e di ottimizzazione dell'efficienza della macchina. Non sempre (e tantomeno in questa sede) è utile addentrarsi nei livelli di maggior dettaglio.

Si riterranno noti in questo articolo i concetti e gli algoritmi già pubblicati da Mondo Digitale sull'aritmetica ([1, 2, 3] e bibliografia ivi) e si metteranno in evidenza alcuni concetti. Il primo è che un calcolatore definisce per le sue operazioni una aritmetica (o più aritmetiche), intesa come la classe dei numeri che esso tratta (interi, reali,...), il numero di bit assegnati alla rappresentazione di un dato e la tecnica di rappresentazione (binaria o decimale, con segno o senza segno ecc.); in ogni caso, con n bit è possibile rappresentare solo un numero finito di numeri e si pongono almeno due problemi: l'intervallo cui appartiene la classe e, per i numeri reali, l'approssimazione (Von Neumann chiamò pseudoreali tali numeri).

Una macchina aritmetica deve trattare, oltre la tecnica o l'algoritmo per calcolare il risultato, anche le *eccezioni* che eventualmente si verificano nell'esecuzione dell'operazione; fra queste le principali sono:

- l'*overflow*, cioè il tentativo di rappresentare un numero esterno all'intervallo;
- l'*underflow*, cioè un numero che, per l'approssimazione, diventa 0 pur non essendolo.

Gli algoritmi per l'esecuzione manuale delle operazioni fondamentali sono tutti sviluppati sui numeri naturali (interi non negativi) rappresentati in aritmetica decimale pesata e, quindi, da questi estesi a insiemi numerici più ampi.

Per esempio, il classico algoritmo per l'addizione manuale di numeri (addizione per cifre, valutazione del riporto ecc.) nasce per i numeri naturali, si estende poi ai numeri positivi "con la virgola" allineando le cifre di egual peso e trattando il numero come un intero formato dalla parte intera e da quella decimale e

si estende ancora ai numeri relativi introducendo le regole per il trattamento del segno e del modulo: in ogni caso l'operazione avviene su numeri naturali che "rappresentano" i numeri della classe da trattare.
Il procedimento che si adotta per realizzare le macchine aritmetiche è simile: si realizza l'aritmetica dei numeri naturali, che opera su interi X, con $0 \le X < M$. Si trattano poi classi di numeri x diverse (per esempio, i numeri relativi) e ciascun elemento x si riconduce con apposita corrispondenza $X = r(x)$ in un numero naturale X (o due nel caso della virgola mobile) che lo rappresenti; le operazioni aritmetiche, allora, si effettuano ancora sui numeri naturali X, ma in quanto rappresentativi dell'insieme dei numeri x (numeri relativi, numeri reali ecc.).

2. Una macchina fondamentale: l'addizionatore modulo-*M*

L'aritmetica dei numeri naturali assume quindi grande importanza perché tutte le altre aritmetiche si rifanno ad essa. L'addizione, (o la somma algebrica), poi, è componente fondamentale di tutte la operazioni aritmetiche e quindi l'addizionatore dei numeri naturali è componente fondamentale di tutta l'aritmetica.
I numeri naturali sono detti "*unsigned*" in molti calcolatori e sono i numeri interi x con $0 \le x < M$. Per l'aritmetica binaria e per numeri rappresentati su n bit si ha $M = 2^n$. Sul piano astratto (ma, come si vedrà, anche molto concreto), si introduce una macchina detta *addizionatore modulo-M* (*add-mod-M*)[1].
Al di là del formalismo, la macchina "add-mod-10" è quella che la nostra mente ha usato fin dalle elementari per effettuare le addizioni: essa rappresenta, infatti, il meccanismo mentale con il quale, per esempio, si fa «8 + 7 (addendi) + "1 che riportavo" (riporto entrante, *carry-in*)» è 16; scrivo 6 (somma-modulo 10) e riporto 1 (*carry out*)». La macchina definita non è altro che la generalizzazione a un qualsiasi modulo M dell'operazione: essa ha in input i due addendi X, Y e il carry-in r, fornisce in uscita la somma in modulo S e il carry-out R.
L'adder modulo-2^n è l'addizionatore per numeri naturali rappresentati su n bit; per esso è $r = 0$ (ma volendo usare la macchina anche in applicazioni speciali è bene lasciare r qualsiasi) ed R coincide con l'indicatore di overflow. Si dirà in sintesi : "l'*addizionatore di numeri naturali è l'add-mod-M, con l'overflow coincidente con il carry-out*".
Per passare dal *cosa* l'addizionatore fa al *come* esso è fatto, si può dire che, così come la tradizionale addizione su n cifre decimali si effettua attraverso n "addizioni modulo-10", un addizionatore mod 2^n si può realizzare attraverso n add-mod-2 (detti anche *full adder*, *FA*). Ne deriva lo schema di figura 1, realizzato con n stadi di *FA*, ciascuno dei quali, dedicato a una coppia di cifre

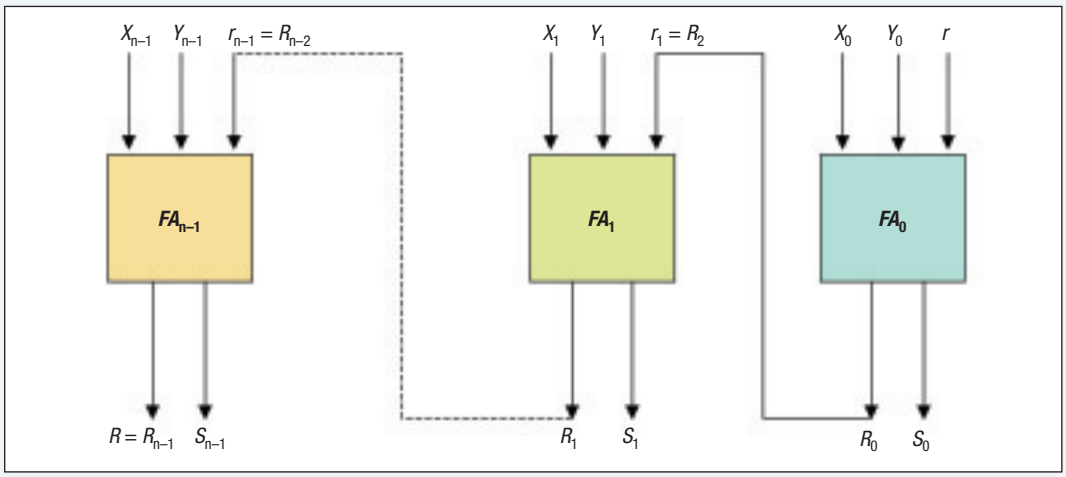

FIGURA 1
Architettura di un addizionatore parallelo

[1] Le macchine informatiche si prestano a definizioni formali nei cosiddetti *Hardware Description Languages*, che non lasciano equivoci sul tipo di dati trattati e sugli algoritmi usati e che sono alla base della realizzazione fisica dei circuiti. Qui si è preferito una presentazione più "leggera"; per ritrovare la definizione formale delle macchine trattate si veda [5].

(bit) X_i, Y_i, riceve come carry-in il carry-out dello stadio di peso immediatamente meno significativo. L'unica differenza fra questa rete e il processo manuale è, oltre alla base di numerazione, che quello manuale opera disponendo di un unico addizionatore di cifra (la mente) e usandolo in tempi successivi per le diverse addizioni di cifra, mentre le macchine moderne usano in parallelo *n* full adder collegati come in figura (è anche possibile una soluzione seriale, ma sarebbe lenta!).
Ma come è fatto l'addizionatore-modulo-2? Per comprenderlo occorre conoscere un po' di algebra di Boole e di elementi di reti logiche. Chi non avesse tali nozioni può anche saltare la coda di questo paragrafo: non toglie molto alla comprensione del tutto. Le tabelline che in [2] illustrano le regole dell'addizione binaria si possono trasformare in tabelle di verità e da queste si può trarre che le funzioni booleane *S* (somma) e *R* (carry out) si realizzano con la rete logica descritta dalle seguenti equazioni booleane[2]:

$$S = \bar{x} \cdot \bar{y} \cdot r + \bar{x} \cdot y \cdot \bar{r} + x \cdot \bar{y} \cdot \bar{r} + x \cdot y \cdot r$$

$$R = x \cdot y + x \cdot r + y \cdot r$$

3. Una macchina per accelerare la propagazione dei riporti

Anche se nella struttura parallela dell'addizionatore tutti i full adder iniziano simultaneamente a operare, il tempo per la conclusione dell'operazione è condizionato dal fenomeno della propagazione dei riporti: l'uscita di uno stadio è corretta solo quando lo è il suo riporto entrante; detto ε il tempo di propagazione del riporto attraverso uno stadio, il tempo di ritardo complessivo potrebbe raggiungere il valore di $n\varepsilon$ se il riporto dovesse propagarsi fra tutti gli *n* stadi [5]. Per calcolare i riporti si può allora usare una tecnica alternativa (anche se più costosa) a quella della propagazione attraverso gli stadi: calcolarli direttamente in funzione dei bit da sommare in un'apposita macchina detta *anticipatore di riporti* (carry look ahead). A tale scopo, si consideri che il carry-out è 1 in due distinte circostanze:
i) viene generato nello stadio perché è $x = y = 1$ (indipendentemente dal carry-in) oppure
ii) essendo *x* e *y* l'uno o e l'altro 1, il carry-in viene "propagato" nel carry-out.
Posto allora per ciascuno stadio (siamo ancora alle espressioni booleane!): $\boldsymbol{G = x \cdot y}$ (condizione di generazione del riporto: *x* ed *y* entrambi 1) e $\boldsymbol{P = x \oplus y}$ (condizione di propagazione del riporto: uno dei due o, l'altro 1), si ha $\boldsymbol{R = G + P \cdot r}$ e l'anticipatore, sviluppando iterativamente tale formula, potrebbe calcolare i carry-out in funzione dei soli *G* e *P* dei singoli stadi[3].
Purtroppo, la complessità circuitale cresce significativamente al crescere del numero di bit e si adotta allora una soluzione di compromesso: si suddivide l'addizionatore in gruppi di bit e si realizza un anticipatore per ciascun

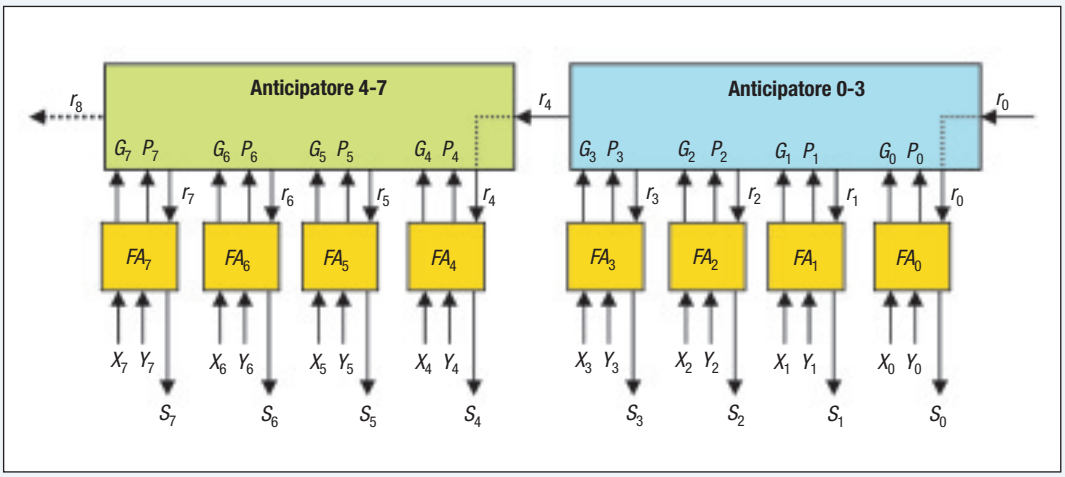

FIGURA 2
Adder con anticipatore di riporti

[2] Qui e nel seguito, nelle espressioni logiche useremo i simboli "·" per la funzione booleana AND, "+" per la OR, "⊕" per la XOR e il segno su una variabile per la NOT.

[3] Si avrebbe per esempio $r_4 = R_3 = G_3 + G_2 P_3 + G_1 P_3 P_2 + G_0 P_3 P_2 P_1 + r_0 P_3 P_2 P_1 P_0$.

gruppo, come esemplificato in figura 2, ove sono posti in evidenza i primi due anticipatori, ciascuno di 4 bit, di una catena che, per esempio, si sviluppa fino a $n = 64$.
Si noti che i riporti entranti in ciascun adder sono definiti quasi simultaneamente dagli anticipatori, mentre la propagazione dei riporti avviene ora tra questi ultimi. Per migliorare l'accelerazione, si potrebbe sostituire la propagazione tra gli anticipatori con un ulteriore livello di anticipatori.

4. Addizionatore in complementi a 2

Si ricorda che, nel caso dei complementi a 2, la relazione tra numero da rappresentare x e numero rappresentato X è:

$$x = -X_{n-1}\,2^{n-1} + \sum_{i=0}^{n-2} X_i\,2^i$$

([2], ma ivi i simboli sono invertiti), il bit X_{n-1} è il bit-segno e gli altri sono i bit-cifra. Si può in effetti anche dimostrare che, leggendo X come un numero naturale (tutti bit-cifra), X è il resto-modulo-2^n di x (cioè il resto della divisione intera per difetto di x per 2^n, come in [3]):

$$X = |x|_M$$

Si assumerà questa come regola di trasformazione di un intero relativo x nel numero naturale X, con $-M/2 \le x < M/2$ e $0 \le X < M$. Il vantaggio di questa rappresentazione è contenuto nella proprietà dei resti in modulo:

$$|x \pm y|_M = \big||x|_M \pm |y|_M\big|_M$$

che, letta alla luce della rappresentazione per complementi, afferma "*La rappresentazione della somma* (*algebrica*) *di x e y si ottiene come somma* (*modulo-M*) *delle rappresentazioni di x e y*".
In altri termini, si afferma che il medesimo addizionatore-modulo-*M* che realizza la somma *senza segno* $X + Y$ realizza anche la somma *con segno* $x + y$ corrispondente: la macchina "non sa" se sta operando sugli uni o sugli altri, ma solo l'uomo che ne interpreta i risultati.
Purtroppo, ciò è vero per il calcolo della somma, ma non per l'individuazione di un eventuale overflow, che nel caso di addendi *con segno* non è più dato, come per quelli senza segno, dal riporto uscente. Per calcolare l'overflow invece occorre far riferimento alle regole della rappresentazione del numero ed alla considerazione che la somma è inclusa nell'intervallo $[-M, M)$; un overflow positivo (derivante dai due addendi entrambi positivi) è incluso allora nell'intervallo $[M/2, M)$, la cui rappresentazione in resti-modulo-*M* è quella di un numero negativo (per una dimostrazione grafica si veda [4]); analogamente, un overflow negativo genera una somma modulo-*M* che è la rappresentazione di un numero positivo. Si ha dunque in generale:

$$\textit{overflow} = x > 0 \textbf{ and } y > 0 \textbf{ and } s < 0 \textbf{ or } x < 0 \textbf{ and } y < 0 \textbf{ and } s > 0$$

o, con riferimento al bit- segno e usando l'algebra di Boole:

$$\textit{overflow} = \overline{X_{n-1}}\;\overline{Y_{n-1}}\;S_{n-1} + X_{n-1}\;Y_{n-1}\;\overline{S_{n-1}}$$

Questa considerazione è assunta in concreto da molti calcolatori reali che posseggono un unico addizionatore per l'aritmetica con segno e senza segno, ma che costruiscono due distinti segnali di overflow: R_{n-1} per numeri senza segno e Overflow per numeri con segno: è cura del programmatore (in linguaggio *assembler*) o del compilatore (in linguaggio ad alto livello) interrogare l'uno o l'altro segnale a seconda dell'aritmetica con la quale sta operando.

5. La macchina "sottrattore-modulo-*M*"

Un addizionatore modulo-*M* può anche effettuare semplicemente la sottrazione $X - Y$: basta che calcoli l'opposto di Y e lo addizioni a X; ma, usando i complementi, l'opposto di Y è $2^n - Y$ e questa sottrazione si può eseguire molto banalmente. Per fissare le idee, si faccia un'analogia con l'aritmetica decimale: il complemento a 10.000 di 1834 è 8166, che si ottiene facilmente effettuando il complemento a 9 "cifra a cifra" (8165) e aggiungendovi quindi 1. In binario si può, dunque, fare il

complemento a 1 "bit a bit" e, quindi, aggiungere 1 al risultato.
Un sottrattore binario si realizza, quindi, sommando a X il complemento bit a bit di Y e ponendo $r = 1$ (è stato detto precedentemente che conveniva lasciare comunque qualsiasi il riporto entrante dell'addizionatore).

6. Il moltiplicatore

Se si opera in un'aritmetica in modulo $M = b^n$, in analogia all'addizionatore, si può definire un *moltiplicatore modulo-M* il quale, dati i due fattori X, Y (entrambi minori di M) calcola due numeri ancora minori di M: $P = |X \cdot Y|_M$ (resto in modulo del prodotto) e $Q = [(X \cdot Y)/M]$ (parte intera del rapporto fra il prodotto ed M). Si noti che tutti i dati trattati dalla macchina hanno n cifre (bit per $b = 2$), mentre il prodotto è $Z = Q \cdot M + P$ e si ottiene dal numero formato dalle cifre (bit per $b = 2$) di Q seguite da quelle di P. Per fissare le idee, si ha per esempio, per $b = 10$ e $n = 4$:

X	Y	Q	P	Z
0025	0152	0000	3800	0000 3800
2222	4444	0987	4568	0987 4568

Così come l'add-mod per l'addizione, il molt-mod è l'apparecchiatura fondamentale per le moltiplicazioni. In particolare, per un'aritmetica di numeri naturali in $[0, M)$, P è il prodotto e Q è un indicatore di overflow se diverso da 0: "*il moltiplicatore di numeri naturali è molt-mod-M, con risultato P ed overflow coincidente con la condizione* $Q \neq 0$". Per fissare le idee, si ha per esempio:

X	Y	Q	P	Prodotto	Ovflo
0025	0152	0000	3800	3800	NO ($Q = 0$)
2222	4444	0987	4568	4568	SI ($Q \neq 0$)

Per la rappresentazione delle mantisse dei numeri in virgola mobile, i calcolatori usano anche una "aritmetica dei frazionari": per $b = 2$, la stringa di n bit che rappresentava un intero viene interpretata come divisa per 2^n, sicché i numeri senza segno cadono nell'intervallo $0 \leq X < 1$ e quelli con segno in $-1/2 \leq x < 1/2$. Mentre per l'addizione l'aritmetica dei frazionari coincide con quella degli interi, per la moltiplicazione, invece, detti $x' = X/M$, $y' = Y/M$ i fattori frazionari si ha $z' = x' y' = XY/M^2 = Q/M + P/M^2$ e, dunque, non si ha mai overflow, le cifre più significative del prodotto sono quelle di Q e, quindi, il prodotto rappresentato su n cifre è Q (e non P come nel caso degli interi) e P/M^2 è l'approssimazione del prodotto.
Il moltiplicatore, allora, deve fornire prodotto e approssimazione e si ha: "*il moltiplicatore di numeri frazionari è molt-mod-M, con risultato Q e approssimazione P (fatti salvi i fattori di scala)*". Per fissare le idee, si ha per esempio:

X	Y	Q	P	Prodotto	EPS
0025 (,0025)	0152 (,0152)	0000	3800	0000 (,0000)	3800 (,00003800)
2222 (,2222)	4444 (,4444)	0987	4568	0987 (,0987)	4568 (,00004568)

Limiti di spazio non consentono di presentare il molt-mod-M che opera in complementi (vedi comunque [2]) né la "macchina moltiplicatore" che, sulla base di quanto detto, realizza l'algoritmo per la moltiplicazione considerando bit per bit il moltiplicatore (analogamente alla moltiplicazione in decimale cifra per cifra).

7. Macchine in virgola mobile

In virgola mobile (*floating point*) un numero reale x è rappresentato mediante una coppia di numeri (M, E), con:

$$x = M \cdot b^E \pm \varepsilon$$

dove M è la *mantissa*, E l'*esponente*, b (= 2, 8, 10 o 16) è la base di numerazione, ε l'approssimazione della rappresentazione [4]. La coppia non è unica; per esempio, per $b = 10$ e supponendo la mantissa espressa su 5 cifre, il numero π (= 3,14159265...) potrebbe essere

espresso, con approssimazioni diverse, come $0{,}00314 \times 10^3$, $0{,}03142 \times 10^2$, $0{,}31416 \times 10^1$; di queste, l'ultima è ovviamente la più precisa e viene detta normalizzata: la rappresentazione normalizzata di un numero è quella che inizia con una cifra non nulla e, sempre che sia possibile, un numero va memorizzato nella forma normalizzata.

Le operazioni in virgola mobile trattano separatamente mantissa ed esponente; vengono qui presentate alcune regole esemplificando in aritmetica decimale.

- Per l'addizione è necessario dapprima rendere eguali i due esponenti e poi effettuare l'addizione:
  $0{,}5211 \times 10^3 + 0{,}1000 \times 10^2 = 0{,}5211 \times 10^3 + 0{,}0100 \times 10^3 = 0{,}5311 \times 10^3$.
- Un overflow di mantissa è "recuperato" se si scala a destra la mantissa e si aumenta di 1 l'esponente:
  $0{,}5000 \times 10^{-4} + 0{,}5000 \times 10^{-4} = 1{,}0000 \times 10^{-4}$ (overflow di mantissa) $= 0{,}1000 \times 10^{-3}$ (overflow recuperato).
- Il vero overflow è un "overflow di esponente"; se, per esempio, il valore massimo dell'esponente è 9 si ha:
  $0{,}5000 \times 10^9 + 0{,}5000 \times 10^9 = 1{,}00000 \times 10^9$ (overflow di mantissa) $= 0{,}10000 \times 10^{10}$ (overflow di esponente).
- Una mantissa risultante da un'operazione può essere normalizzata se la si scala a sinistra diminuendo di 1 l'esponente:
  $0{,}5432 \times 10^{-2} \times 0{,}1000 \times 10^{-2} = 0{,}05432 \times 10^{-4} = 0{,}5432 \times 10^{-5}$.
- Una moltiplicazione si effettua calcolando il prodotto delle mantisse e la somma degli esponenti:
  $0{,}5000 \times 10^1 \times 0{,}5000 \times 10^1 = 0{,}2500 \times 10^2$.
- Oltre all'overflow, l'aritmetica in virgola mobile deve trattare l'*underflow*, che si ha quando, a causa dei limiti della rappresentazione, il risultato di un'operazione deve essere approssimato allo zero: se -9 è il valore minimo dell'esponente:
  $0{,}1000 \times 10^{-9} \times 0{,}1000 \times 10^{-9} = 0{,}0100 \times 10^{-18}$ = underflow.

Quanto esemplificato per $b = 10$ vale anche per $b = 2$; lo standard oggi in voga, lo IEEE 754 [6], suggerisce che il numero x sia rappresentato dalla tripla (s, f, e) con $s \in \{0, 1\}$ segno, $0 \leq f < 1$ numero frazionario, e intero relativo, con

$$x = (-1)^s (1 + f) \cdot 2^e \pm \varepsilon$$

e fissa i limiti minimo e massimo per l'esponente nonché il numero di bit per la mantissa. La rappresentazione della mantissa è in segno e modulo e, in particolare, per parole di 32 bit ne occupa 24 (compreso il segno), mentre l'esponente ne occupa 8. La particolare notazione dello standard ($1 + f$ invece di f) fa risparmiare un bit nella memorizzazione di un dato.

Gli algoritmi e quindi, le macchine in virgola mobile si traggono semplicemente dalle regole generali prima elencate; si mettono soltanto qui in evidenza alcune macchine che, oltre a quelle per le operazioni in virgola fissa su mantisse ed esponenti, ne costituiscono le macchine elementari componenti:

- *float-shif- left* (*X*): scala a sinistra la mantissa e decrementa di 1 l'esponente;
- *float-shift-right* (*X, eccezione*): scala a destra la mantissa ed incrementa di 1 l'esponente; se questo è già pari al massimo consentito, pone *eccezione* = **true**;
- *normalizza* (*X, eccezione*): normalizza *X* (mediante float-shift-left) e, in caso di underflow, pone *eccezione* = **true**.

8. Conclusioni

Limiti di spazio impediscono di approfondire ulteriormente il tema delle macchine aritmetiche, che qui si è trattato soprattutto presentando un metodo basato su regole aritmetiche. Da queste si evincono due conclusioni fondamentali. La prima è che la struttura delle macchine deriva dall'adozione di sistemi di rappresentazione "posizionali a cifre pesate"; l'impiego del sistema binario, pur rilevante ai fini della realizzazione dei circuiti, non condiziona gli algoritmi, che sono di carattere del tutto generale rispetto alla base di numerazione.

La seconda conclusione è che, da un punto di vista di principio, nulla cambia (a parte l'efficienza, e non è poco!) se le macchine sono realizzate in *software* o in *hardware*, tanto è che una buona progettazione dell'hardware parte dalla descrizione in software dell'algoritmo mediante gli *Hardware Description Languages* (HDL).

Bibliografia

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

[2] Ciminiera L.: Le operazioni aritmetiche. *Mondo Digitale*, Anno III, n. 10, giugno 2004, p. 77-81.

[3] Stefanelli R.: L'aritmetica dei residui ed il suo uso per la realizzazione di unità aritmetiche specializzate particolarmente veloci. *Mondo Digitale*, Anno III, n. 11 settembre 2004, p. 74-77.

[4] Fadini B., Savy C.: *Fondamenti di Informatica*.I (2ª edizione), Liguori, 1997.

[5] Fadini B., De Carlini U.: *Macchine per l'elaborazione dell'informazione*.(2ª edizione), Liguori, 1995.

[6] IEEE Computer Society: *IEEE Standard for Binary Floating-Point Arithmetic*. IEEE Std 754, 1985.

Si veda anche la bibliografia in [1, 2 e 3].

Bruno Fadini è professore ordinario di Calcolatori elettronici presso la Facoltà di Ingegneria dell'Università di Napoli Federico II. È membro dell'Accademia Pontaniana, ed attualmente è Direttore del CINI (Consorzio Interuniversitario Nazionale per l'Informatica), membro della Commissione Informatica e Telematica dell'Ateneo Federico II, Presidente del Comitato Tecnico-scientifico del CRIAI (Consorzio di ricerca campano), membro del Comitato di esperti del MIUR per le Università telematiche.

Roberto Canonico è ricercatore presso la Facoltà di Ingegneria dell'Università di Napoli Federico II, dove è attualmente docente per supplenza di Calcolatori Elettronici e Reti Logiche. Laureato in Ingegneria Elettronica nel 1995, ha conseguito il titolo di dottore di ricerca in Ingegneria Elettronica ed Informatica nel 2000. Nel 2001 è stato *Visiting Research Associate* presso la Lancaster University (Gran Bretagna). È membro dell' IEEE Computer Society e dell'Association for Computing Machinery (ACM).