UNIVERSITÀ DEGLI STUDI DI PISA

ISTITUTO DI SCIENZE DELL'INFORMAZIONE

Franco Preparata

RAPPRESENTAZIONE DELL'INFORMAZIONE
ELEMENTI DI ARITMETICA DEI CALCOLATORI

Note per il Corso di

Teoria ed Applicazioni delle Macchine Calcolatrici

ANNO ACCADEMICO 1972 - 73

INDICE

La rappresentazione dell'informazione
Elementi di aritmetica dei calcolatori

1. Rappresentazione dell'informazione

Il problema della rappresentazione dell'informazione è di fondamentale importanza per l'attività di comunicazione tra esseri intelligenti. Si è così abituati a far uso di opportune rappresentazioni delle informazioni, da considerarle, in certo modo come scelte naturali. In effetti, se si riflette brevemente, ci si accorge che la comunicazione - cioè la trasmissione di informazione - fa necessariamente uso di "simboli", cioè di segni speciali, per mezzo dei quali si rappresentano i "significati" da trasmettere. La distinzione tra "simbolo" e "significato" è fondamentale: essa è particolarmente evidente quando si considerano le nozioni di "cifra" e "numero": la prima è un simbolo mentre il secondo è una nozione astratta. La rappresentazione o codifica è appunto la condizione che stabilisce la corrispondenza tra simboli e significati: tale convenzione è essenziale perchè la comunicazione possa aver luogo.

In questa sede ci occuperemo solo di simboli e delle loro capacità di rappresentazione. Ovviamente la scelta dei simboli è arbitraria. Un criterio ragionevole per la loro scelta è che siano facilmente distinguibili e producibili. Queste proprietà sono soddisfacentemente possedute da insiemi di simboli ben noti, quali l'insieme delle cifre arabe (cioè, le cifre da 0 a 9) e delle lettere dell'alfabeto latino (cioè, le lettere A,B,C,...,Z). D'ora in poi, per uniformità, chiameremo alfabeto un qualunque insieme di simboli adottati per rappresentare l'informazione.

E' abbastanza naturale che la necessità di rappresentare dati per un calcolatore porti alla scelta di un alfabeto differente da quelli cui si è abituati nella pratica ordinaria. Si è trovato che un alfabeto binario, cioè quello che consiste di due soli simboli, ad esempio 0 e 1, è il più adatto alla rappresentazione di dati per un calcolatore elettronico, in quanto questi due simboli possono essere associati con due stati di svariati componenti fisici, stati che sono producibili con

grande affidabilità e facilmente distinguibili. Queste coppie di stati sono, ad esempio: la conduzione o la non-conduzione di un transistore, i due stati di magnetizzazione residua di un nucleo di ferrite, la presenza o l'assenza di una perforazione in una certa posizione di un nastro di carta, ecc.. Una volta adottato l'alfabeto binario (adozione quasi universale per i calcolatori elettronici, se si ignorano alcuni tentativi poco fruttuosi di adottare un alfabeto ternario), è utile sottolineare il fatto che i dati sono talvolta rappresentati direttamente (come nel caso dei numeri), e talvolta indirettamente (cioè rappresentandone, per mezzo dell'alfabeto binario, una rappresentazione simbolica in altro alfabeto, come nel caso dell'informazione alfabetica).

In termini un poco più astratti, il problema della codifica o rappresentazione in binario dell'informazione si può formulare come segue. Si abbia un insieme di s dati distinti, $A = \{a_1, a_2, \ldots, a_s\}$; si stabilisca una corrispondenza tra ciascun elemento a_i di quest'insieme e una sequenza di simboli appartenenti all'alfabeto $\{0,1\}$. Questa corrispondenza viene frequentemente chiamata il <u>codice</u> della rappresentazione. Per semplicità, ci limiteremo a considerare codici le cui sequenze di simboli (chiamate <u>configurazioni</u>) hanno lunghezza costante k cioè consistono di k simboli. Una condizione necessaria per l'esistenza di un codice è allora che vi siano almeno tante configurazioni quanti sono i dati distinti da rappresentare. Poichè è facile riconoscere che si hanno 2^k diverse configurazioni di k cifre binarie, si ha la condizione

$$2^k \geq s \quad . \qquad (1)$$

Ad esempio se A è l'insieme delle 26 lettere dell'alfabeto inglese, allora $k \geq 5$, poichè $2^4 = 16 < 26 < 32 = 2^5$: occorrono cioè almeno 5 cifre binarie. Una volta che la condizione necessaria (1) sia soddisfatta, il codice è una delle possibili scelte, il numero delle quali è molto grande anche per s relativamente piccolo. E' facile riconoscere che questo numero è dato da $2^k(2^k-1)(2^k-2)\ldots(2^k-s+1) = 2^k!/(2^k-s)!$, poichè la rappresentazione di a_1 può essere scelta in 2^k modi, quella di a_2 in (2^k-1) modi,..., quella di a_s in (2^k-s+1) modi.

Nel caso di informazione non numerica la scelta del codice non è affatto critica, ma è sostanzialmente arbitraria. Ciò è dovuto al fatto che l'elaborazione di informazione non numerica è basata principalmente sul "riconoscimento" di determinate configurazioni Non così nel caso dell'informazione numerica, la cui elaborazione consiste frequentemente in manipolazioni di tipo aritmetico, quali addizione, moltiplicazione, ecc.. Poichè è abbastanza ovvio che la scelta del codice influenzerà la complessità delle unità fisiche preposte alla realizzazione delle manipolazioni in questione, è naturale che si ricerchino codici altamente strutturati al fine di semplificare le apparecchiature. A questo tipo di problemi è dedicato il resto di questo capitolo.

Si può saltare

2. La rappresentazione dei numeri

E' facile convincersi, a patto di rinunciare ad un ormai radicato e pressochè meccanico atteggiamento verso la rappresentazione e manipolazione di numeri, che il sistema di numerazione arabica ha carattere posizionale. In altri termini, le cifre hanno diverso valore (tecnicamente, <u>significatività</u>) a seconda della posizione che occupano nella sequenza di cifre che rappresenta un numero N. Ad esempio, l'intero N rappresentato in decimale dalla sequenza 252 ha il valore numerico $N = 2\times10^2 + 5\times10^1 + 2\times10^0$. Si noti che la stessa cifra 2 appare nella posizione delle centinaia ed in quella delle unità. In generale un intero N, rappresentato in <u>base</u> decimale dalle n cifre $d_{n-1}\ d_{n-2}\ \ldots\ d_0$ ha il valore numerico

$$N = \sum_{j=0}^{n-1} d_j\ 10^j \quad ,$$

dove ognuna delle d_j può assumere uno dei valori 0,1,2,...,9. Malgrado la nostra ancestrale consuetudine, non c'è nessuna speciale ragione che porti a scegliere come base l'intero 10, di modo che le sue successive potenze definiscano la significatività delle posizioni di cifra. In verità, ci si può facilmente convincere che la scelta di un qualsiasi valore $r > 1$ sarebbe stata del tutto adeguata per rappresentare i numeri interi (e, difatti, il calcolatore digitale ci ha condotti a scegliere una base che, essendo la più piccola possibile, risulta la più adeguata per realizzare apparecchiature economiche ed affidabili).

Una volta assunto l'intero r come base (o "radice") per la rappre-

sentazione dei numeri, il valore numerico dell'intero rappresentato, in base r, dalla sequenza di n cifre $N \equiv b_{n-1}\ b_{n-2} \cdots b_0$ si ottiene immediatamente come risultato del calcolo

$$N = \sum_{j=0}^{n-1} b_j\ r^j \ . \qquad (2)$$

Si noti che comunque si scelga la base r e comunque si fissino i valori (interi) dei coefficienti b_j, il membro destro della (2) definisce sempre un valore (intero) N. Tuttavia non tutte le scelte di r e dei possibili valori dei coefficienti b_j conducono ad un sistema di rappresentazione soddisfacente. Infatti richiederemo che, per un qualunque intero N, la rappresentazione

1) esista

2) sia unica.

Entrambi questi requisiti sono soddisfatti se i b_j possono assumere i valori 0,1,...,r-1. Questo risultato è enunciato senza dimostrazione. Ogni insieme di possibili valori per i b_j che includa l'insieme 0,1,...,r-1 assicura ancora l'esistenza della rappresentazione, ma non la sua unicità. In altri termini ad un dato intero può corrispondere <u>più di una</u> rappresentazione. Tali sistemi sono detti <u>ridondanti</u> (poichè le rappresentazioni distinte sono più dei valori numerici distinti) e sono talvolta adottati con scopi particolari (maggiore sicurezza e/o velocità di funzionamento ecc.).

ESEMPIO Il valore numerico dell'intero, rappresentato in base 5 da $N \equiv 423$, è $N = 3\times5^0 + 2\times5^1 + 4\times5^2 = 113$. L'ultimo numero, "centotredici", è rappresentato in decimale. Si tenga presente, per la comprensione del seguito, che, d'ora in poi, mentre un numero N potrà essere rappresentato in una base qualsiasi r, il suo valore numerico sarà rappresentato, per convenienza e uniformità, sempre in base 10. Ad esempio, 423 in base 5 ha valore numerico 113: sia chiaro, inoltre, che mentre 113 viene letto "centotredici", 423 deve esser letto "quattro, due, tre" e che leggerlo "quattrocentoventitre" corrisponderebbe ad un grave errore concettuale. Per chiarezza, la base r di rappresentazione viene indicata appendendo r come indice alla rappresentazione stessa, cioè 423_5 è la rappresentazione in base 5 del numero la cui rappresentazione decimale è 113_{10}. Non useremo l'indice r quando la base risulti chiara dal contesto.

Meno ovvia, invece, è l'operazione inversa, cioè la determinazione della rappresentazione in base r di un intero N. Qualora l'intero N sia

assegnato mediante il suo valore numerico, cioè, sia espresso nella ordinaria numerazione in base 10 direttamente riflessa nel linguaggio comune, potremo parlare di "conversione da base 10 a base r". Questa conversione, comunque, è da riguardare come un caso speciale di una più generale "conversione da base s a base r". Possiamo effettuare tale conversione in due modi.

1) <u>Procedimento di conversione per divisione</u>. E' un algoritmo iterativo che genera a ciascun ciclo una cifra della rappresentazione di N in base r. Il diagramma di flusso dell'algoritmo è dato in figura 1

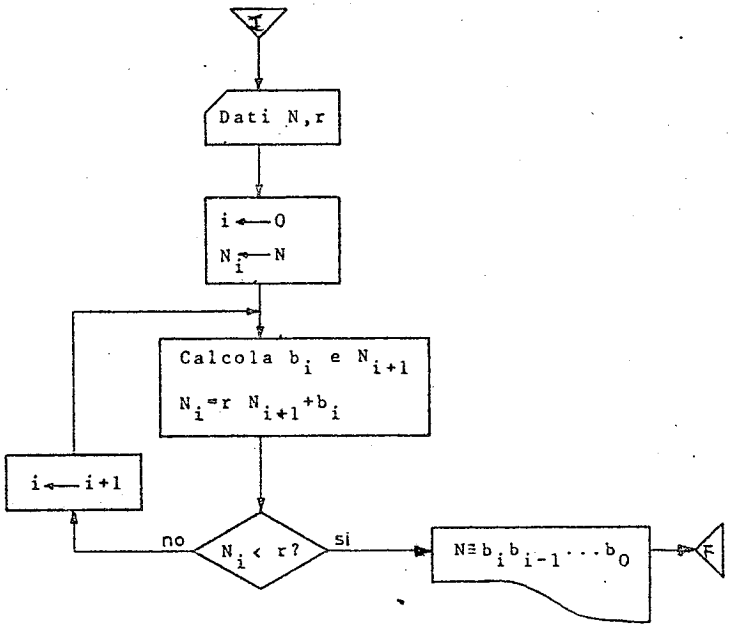


Figura 1.- Diagramma di flusso dell'algoritmo di conversione per divisione.

L'algoritmo è caratterizzato dall'indice i e dalla variabile corrente N_i, cioè l'algoritmo calcola iterativamente la successione $N_0, N_1, N_2, \ldots$ Come inizializzazione si pone i=0 e $N=N_0$. Il passo centrale dell'algoritmo è il calcolo di N_{i+1} e di b_i come quoziente e resto, rispetti-

vamente, della divisione avente N_i per dividendo e r per divisore[1]. Si ha che a ciascun passo $N_{i+1} < N_i$ (se il divisore è maggiore di 1, il quoziente è minore del dividendo), per cui vi è un valore minimo i' di i tale che $N_{i'} < r$: a questo punto la divisione fornisce $N_{i'+1}=0$ e $b_{i'}=N_{i'}$, e a ciascuno dei passi successivi avremo $b_{i'+1}=0$, $b_{i'+2}=0$ ecc. Pertanto l'algoritmo dovrà arrestarsi quando si verificherà la condizione $N_i < r$.

E' semplice dimostrare che la sequenza dei resti $b_i\ b_{i-1}\ldots b_0$ rappresenta N in base r. Infatti sostituendo successivamente N_i nell'espressione di N_{i-1} e procedendo a ritroso si ha: (vedi appunti)

$N_i = b_i$

$N_{i-1} = b_i r + b_{i-1}$

$N_{i-2} = (b_i r + b_{i-1})r + b_{i-2} = b_i r^2 + b_{i-1} r + b_{i-2}$

...

$N_0 = N = b_i r^i + b_{i-1} r^{i-1} + \ldots + b_0$.

(1) $N_0 = N_1 \cdot r + b_0$

$N_1 = N_2 \cdot r + b_1$ sostituendo in (1)

$\Downarrow$

$N_0 = N_2 r^2 + b_1 r + b_0$

e così via.

ESEMPIO: Rappresentare in binario l'intero 185.

j	N_j	b_j	j	N_j	b_j
0	185	1	5	5	1
1	92	0	6	2	0
2	46	0	7	1	1
3	23	1	8	0	stop
4	11	1			

e $185_{10} = 10111001_2$.

E' interessante notare che il procedimento discusso fornisce le cifre della rappresentazione a cominciare dalla meno significativa fino ad arrivare alla più significativa (in ordine di significatività crescente).

(1) Come è noto, secondo l'algoritmo euclideo di divisione, dati due interi a e b, si ottengono due unici interi p e r, con $r < b$, tali che

$$a = bp + r.$$

Si può tralasciare la dimostrazione : vuole la mia

2) Procedimento di conversione per moltiplicazione. Sia N l'intero da convertire. Si assume che N abbia la rappresentazione in base r

$$N = b_{n-1} r^{n-1} + b_{n-2} r^{n-2} + \ldots + b_0 \qquad (3)$$

dove i parametri n, b_{n-1}, b_{n-2},..., b_0 sono da riguardare come incogniti.

Ovviamente, il primo parametro da determinare è l'intero n. Si noti allora che, per una qualsiasi scelta dei coefficienti $b_0, b_1, \ldots b_{n-1}$ nell'alfabeto 0,1,...,r-1 si ha[1]

$$\sum_{i=0}^{n-1} b_i r^i \le \sum_{i=0}^{n-1} (r-1) r^i = (r-1) \sum_{i=0}^{n-1} r^i = (r-1) \frac{r^n - 1}{(r-1)} = r^n - 1 .$$

Pertanto con n cifre r-arie si possono rappresentare gli interi da 0 a $r^n - 1$; sarà quindi necessario scegliere n in modo che $r^n - 1 \ge N$, cioè

$$r^n > N . \qquad (4)$$

Di solito, ma non necessariamente, n è scelto in modo che r^n sia la più piccola potenza di r per cui la (4) è valida. Esempio: per N=185 e r=2 dobbiamo scegliere $n \ge 8$, poichè $2^7 = 128 < 185$.

L'algoritmo che stiamo per descrivere è anch'esso iterativo e genera a ciascun ciclo una cifra della rappresentazione di N in base r. Il diagramma di flusso dell'algoritmo è dato in figura 2. L'algoritmo fa uso dell'indice i e dalla variabile corrente R_i. Come inizializzazione l'algoritmo pone i=1 e $R_1 = N$. L'algoritmo consiste di due passi centrali, uno per il calcolo di b_{n-i} e l'altro per il calcolo di R_{i+1}. La comprensione di questi due passi sarà immediata alla luce delle seguenti considerazioni. Espandiamo la (3) come segue:

$$N = b_{n-1} r^{n-1} + \ldots + b_{n-i+1} r^{n-i+1} + b_{n-i} r^{n-i} + \ldots + b_0$$

Moltiplichiamo ora entrambi i membri per r^{i-1} e raggruppiamo convenientemente i termini del secondo membro:

$$r^{i-1} N = (b_{n-1} r^{n+i-2} + \ldots + b_{n-i+1} r^n) + (b_{n-i} r^{n-1} + \ldots + b_0 r^{i-1}).$$

(1) Per convenienza del lettore, si ricorda che la successione $1, r, r^2, \ldots, r^n, \ldots$ è una progressione geometrica di ragione r. Si ha per $r \ne 1$

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1}$$

Questa relazione si dimostra induttivamente.

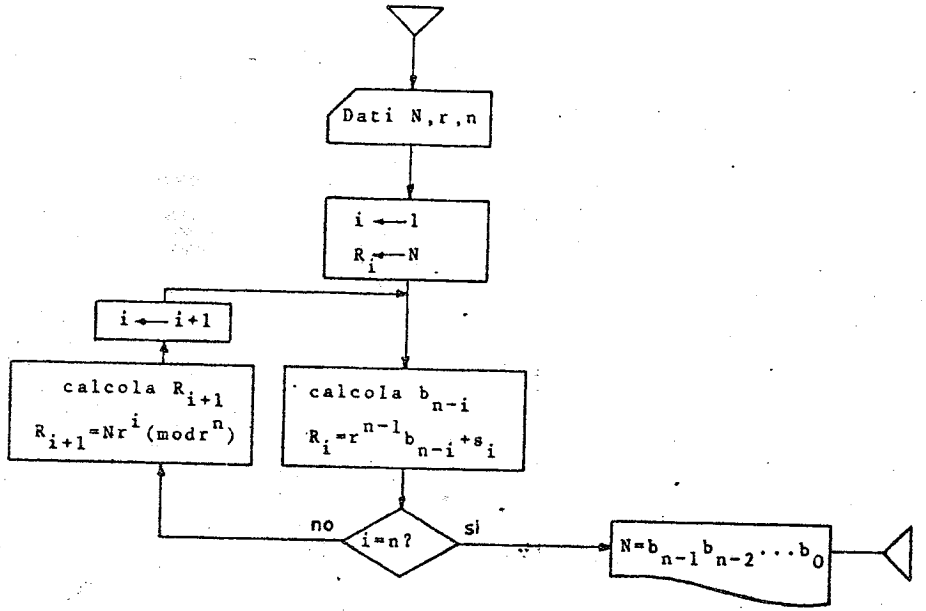


Figura 2.- Diagramma di flusso dell'algoritmo di conversione per moltiplicazione.

Chiaramente ciascuno dei termini entro la prima coppia di parentesi è multiplo di r^n, mentre la somma dei termini entro la seconda coppia di parentesi è $\leq r^{n-1}$, cioè è minore di r^n. Perciò, se dividiamo $r^{i-1}N$ per r^n e chiamiamo $r^{i-1}N(\mathrm{mod}\, r^n)$ il resto di questa divisione, si ha

$$r^{i-1}N(\mathrm{mod}\, r^n) = b_{n-i}r^{n-1}+(b_{n-i-1}r^{n-2}+\ldots+b_0r^{i-1}).$$

Si osservi ora che la somma dei termini in parentesi è minore di r^{n-1}, per cui se dividiamo $r^{i-1}N(\mathrm{mod}\, r^n)$ per r^{n-1} otteniamo il coefficiente b_{n-i} come quoziente. Se quindi definiamo $R_i=r^{i-1}N(\mathrm{mod}\, r^n)$ abbiamo la regola che spiega i due passi cruciali dell'algoritmo:

> "Il coefficiente b_{n-i} è il quoziente della divisione di R_i per r^{n-1}, dove $R_i=r^{i-1}N(\mathrm{mod}\, r^n)$".

Quando dovrà arrestarsi l'algoritmo? Chiaramente, se N è intero l'algoritmo si arresterà col calcolo di b_0, cioè quando i=n, cioè dopo n passi.

In contrasto con il procedimento per divisione questo metodo fornisce le cifre della rappresentazione in ordine di significatività decrescente (la cifra più significativa per prima).

Il calcolo di R_{i+1} come $r^iN(\mathrm{mod}\, r^n)$ è oneroso in quanto comporta una divisione. La seguente osservazione conduce a una meccanizzazione molto semplice del calcolo di R_{i+1}. Si noti che

$$Nr^i=(b_{n-1}r^{n-i-1}+\ldots+b_{n-i}r^n)+(b_{n-i-1}r^{n-1}+\ldots+b_0r^i)$$

per cui

$$R_{i+1}=Nr^i(\mathrm{mod}\, r^n)=b_{n-i-1}r^{n-1}+\ldots+b_0r^i.$$

Si ricordi inoltre (vedi sopra) che:

$$R_i=Nr^{i-1}(\mathrm{mod}\, r^n)=b_{n-i}r^{n-1}+(b_{n-i-1}r^{n-2}+\ldots+b_0r^{i-1}),$$

per cui il resto della divisione di R_i per r^{n-1}, che chiamiamo s_i, è dato da

$$s_i=b_{n-i-1}r^{n-2}+\ldots+b_0r^{i-1}$$

Ma allora è immediato accorgersi che

$$R_{i+1}=r(b_{n-i-1}r^{n-2}+\ldots+b_0r^{i-1})=r\,s_i$$

che consente di enunciare la seguente semplicissima regola per il calcolo di R_{i+1}:

> "R_{i+1} si ottiene moltiplicando per r il resto della divisione di R_i per r^{n-1}, cioè $R_{i+1}=r\left[R_i \bmod r^{n-1}\right]$ "

ESEMPIO: Sia r=3 e N=47. Allora $n \geq 4$ poichè $3^3 < 47 < 3^4$. Il procedimento di conversione viene meccanizzato come segue (notare che con la scelta $n \geq 4$, 47 (modulo 3^n)=47). Scegliendo n=4, $3^{n-1}=3^3=27$ è il divisore:

i	R_i		quoziente b_{n-i}	resto s_i
1	47	=	1 x 27 +	20
2	3x20=60	=	2 x 27 +	6
3	3x6 =18	=	0 x 27 +	18
4	3x18=54	=	2 x 27 +	0

Pertanto $b_3 b_2 b_1 B_0 = 1202$ è la rappresentazione il cui valore numerico è $1x3^3+2x3^2+2=47$.
Si può osservare che l'esecuzione dell'algoritmo può essere accelerata non appena si ottenga $s_i=0$. Infatti, per ogni $j>i$, il passo j-esimo avrebbe la forma

$$3x0=0x27+0$$

per cui possiamo direttamente porre $b_{n-i-1}\, b_{n-i-2}\cdots b_0 = 00\ldots0$.

ESEMPIO: Rappresentare l'intero (decimale) 185 in binario. La più piccola potenza di 2 maggiore di 185 è $256=2^8$: di conseguenza possiamo scegliere n=8. Notando che $2^7=128$ l'algoritmo può essere sviluppato come segue:

i	R_i	b_{n-i}	s_i
1	185=1x128+ 57	b_7	= 1
2	2x57 =114=0x128+114	b_6	= 0
3	2x114=228=1x128+100	b_5	= 1
4	2x100=200=1x128+ 72	b_4	= 1
5	2x72 =144=1x128+ 16	b_3	= 1
6	2x16 = 32=0x128+ 32	b_2	= 0
7	2x32 = 64=0x128+ 64	b_1	= 0
8	2x64 =128=1x128+ 0	b_0	= 1

Si noti che se r=2, si può omettere l'esplicita indicazione dei quozienti, poichè essi possono valere soltanto 0 od 1.

vedere meglio

Fino ad ora ci siamo occupati esclusivamente di interi non-negativi. Se vogliamo considerare la rappresentazione di numeri razionali non negativi, allora osserviamo che il razionale N=a/b sarà la somma di una parte intera $I \geq 0$ e di una parte propriamente frazionaria $F < 1$. Eseguendo la divisione tra a e b abbiamo

$$a = I\, b + e$$

e ponendo e/b=F si ottiene a/b=I+F. La parte I può essere sempre esattamente rappresentata in modo posizionale comunque si fissi la base; lo stesso non si può dire per la parte F, poichè si suppone di poter disporre di un numero finito di cifre. Infatti, in decimale, il razionale 10/3 è rappresentato come

$$3{,}33333\ldots\ldots,$$

(la virgola separa la parte intera dalla parte frazionaria), cioè, possiede una rappresentazione periodica, con periodo diverso da zero, che, se troncata, fornirà un'approssimazione per difetto del razionale assegnato. Potrà anche accadere che, per un razionale N, il periodo sia zero per una certa base (cosicchè la rappresentazione posizionale risulterà esatta) e sia diverso da zero per qualche altra base (vedi esempio nel seguito).

Vedi appunti

Estendendo le precedenti considerazioni sulla rappresentazione posizionale alle potenze negative della base r, si ottiene che il valore numerico della frazione

$$F = b_{-1}\, b_{-2}\, b_{-3}\cdots\cdots \text{ è } \sum_j b_{-j}\, r^{-j}\ .$$

Si noti che le prime j cifre $b_{-1}\, b_{-2}\cdots b_{-j}$ sono la rappresentazione in base r della parte intera di $r^j F$. Infatti ponendo

$$F = b_{-1}r^{-1}+b_{-2}r^{-2}+\ldots+b_{-j}r^{-j}+\ldots\ ,$$

abbiamo

$$r^j F = b_{-1}r^{j-1}+b_{-2}r^{j-2}+\ldots b_{-j}r^0+\ldots$$

e ciò dimostra l'asserzione. Ne segue che il metodo per moltiplicazione è il solo che ci consenta di calcolare la rappresentazione esatta di F, in quanto, producendo le cifre in ordine di significatività decrescente, non pone nessuna limitazione al numero di cifre usate nella rappresentazione.

Poichè $F<1$, n deve essere scelto ≥ 0. Per la base r=2 vi è un vantaggio nello scegliere n=1, poichè in tal caso $r^{n-1}=2^{1-1}=1$, il che consente una semplice meccanizzazione della conversione, come illustrato dall'esempio che segue.

ESEMPIO: Sia F=2/3 e r=2

2xF =4/3=1+1/3 , per cui	$b_{-1}=1$	e 1/3 è il residuo
2x1/3=2/3=0+2/3	$b_{-2}=0$	e 2/3 "
2x2/3=4/3=1+1/3	$b_{-3}=1$	ecc.

da questo punto in poi l'espressione risulta periodica di periodo 2 cioè 2/3=.101010.... .

ESEMPIO: Invece di rappresentare (in modo approssimato) un numero razionale in binario, vogliamo ora convertire da decimale a binario (cioè, il numero razionale sia tale che il suo denominatore sia una potenza di 10). Poniamo F=0,7 e r=2. Allora

2xF =1,4=1+0,4	$b_{-1}=1$
2x0,4=0,8=0+0,8	$b_{-2}=0$
2x0,8=1,6=1+0,6	$b_{-3}=1$
2x0,6=1,2=1+0,2	$b_{-4}=1$
2x0,2=0,4=0+0,4	$b_{-5}=0$
2x0,4=0,8=0+0,8	$b_{-6}=0$

e da questo punto in poi nella rappresentazione si ripete periodicamente la sequenza 1100.

Concludiamo questo paragrafo menzionando la cosiddetta rappresentazione ottale. Si abbia ad esempio un numero binario di 10 cifre $b_9b_8b_7b_6b_5b_4b_3b_2b_1b_0$, il cui valore è ovviamente

$$b_9 2^9+b_8 2^8+b_7 2^7+b_6 2^6+b_5 2^5+b_4 2^4+b_3 2^3+b_2 2^2+b_1 2+b_0.$$

Raggruppiamo, partendo da destra, i termini a tre a tre e poniamo in evidenza le potenze di 2:

$$b_9 2^9+(b_8 2^2+b_7 2+b_6)2^6+(b_5 2^2+b_4 2+b_3)2^3+(b_2 2^2+b_1 2+b_0)$$

Ciascun termine in parentesi è un numero tra 0 e 7; questi termini possono essere considerati come i coefficienti di una rappresentazione in base $2^3=8$, da cui l'aggettivo "ottale". Pertanto la sequenza 100101110001, viene suddivisa in terne come segue: 100 101 110 001, e ciascuna terna interpretata come numero decimale, cioè abbiamo

$$4561_8 = 100101110001 = 2417_{10}$$

La rappresentazione ottale non ha alcun interesse concettuale; l'unico vantaggio è la facile comunicazione di sequenze binarie tra utenti umani.

3. Addizione, sottrazione e rappresentazione dei numeri relativi (positivi e negativi)

In questo paragrafo ci riferiremo esclusivamente a numeri interi. Le regole per addizionare cifre binarie sono date nella tabella I.

	0	1
0	0	1
1	1	10

Tabella I. Tabella per l'addizione binaria.

Estendendo quella che è l'ordinaria procedura per l'addizione in decimale, nell'effettuare un'addizione di due operandi le cifre dello stesso peso (stessa posizione) devono essere sommate come specificato in tabella I, e $1+1=10_2$ chiaramente significa che la somma in quella posizione è 0 e c'è un "riporto" alla successiva posizione. L'introduzione del riporto richiede che sia considerata l'addizione simultanea di tre cifre binarie. Più precisamente, supponiamo che a_j e b_j siano le cifre nella posizione 2^j degli operandi A e B e c_j il riporto dalla posizione 2^{j-1}. Denominando s_j la cifra di peso 2^j della somma e c_{j+1} il riporto alla posizione di peso 2^{j+1}, abbiamo la seguente tabella di addizione:

a_j	b_j	c_j	s_j	c_{j+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabella II

che descrive completamente l'addizione, posizione per posizione, di due operandi binari.

ESEMPIO: Sia A=13=01101 e B=11=01011. Allora

riporti	11110	
A	01101	13
B	+01011	+11
somma	11000	24

In modo simile potremmo descrivere la operazione di sottrazione, estendendo al caso binario la ben nota nozione di "prestito". Tuttavia, la necessità di dispositivi separati per addizione e sottrazione è stata generalmente superata nel progetto dei calcolatori digitali, usufruendo di una speciale rappresentazione dei numeri negativi che permette di ef-

effettuare addizioni e sottrazioni con la stessa unità funzionale.
Prima di considerare la rappresentazione accennata, vogliamo notare che se si fossero adottate due unità separate per addizione e sottrazione, allora una rappresentazione degli interi relativi in "modulo e segno" (dove il modulo, o valore assoluto, è rappresentato come un numero non negativo e il segno è indicato da un bit addizionale) sarebbe perfettamente adeguata: in verità l'addizionatore verrebbe usato per l'addizione di operandi di segno uguale e per sottrazione di operandi di segno diverso, mentre il sottrattore sarebbe usato per l'addizione di operandi di segno diverso e per la sottrazione di operandi di ugual segno. Sottolineiamo esplicitamente che la selezione dell'unità verrebbe a dipendere dai segni degli operandi.

Gli svantaggi suaccennati hanno condotto ad adottare rappresentazioni degli interi relativi chiamate rappresentazioni complementate. Ci occuperemo di due casi importanti, la "complementazione a 2" e la "complementazione a 1".

*

3.1 - La rappresentazione in complemento a 2

La rappresentazione di N è come al solito,

$$\underset{\text{segno}}{b_n} \quad \underbrace{b_{n-1}\, b_{n-2} \cdots b_0}_{\text{parte numerica}}$$

dove, per convenienza, b_n viene designato come "bit di segno" e $b_{n-1}\cdots b_0$ viene designata come "parte numerica". Il valore di N è dato da

$$N = -b_n 2^n + \sum_{j=0}^{n-1} b_j 2^j \qquad (5)$$

cioè, il peso della posizione b_n è ora -2^n. Il massimo numero rappresentabile è $011\ldots1 = (2^n-1)$, mentre il più piccolo è $1000\ldots0 = -2^n$ (cioè, $b_n=1$, il che massimizza la parte negativa della (5), e $b_0=b_1=\ldots=b_{n-1}=0$ che minimizza la parte positiva della (5)).
Di conseguenza l'intervallo di N è $-2^n \le N \le 2^n-1$. Per un numero negativo N si ha $b_n=1$ e poichè per un tale intero $N=-|N|$, la (5) diventa

$$-|N| = -2^n + \sum_{j=0}^{n-1} b_j 2^j ,$$

oppure

$$\sum_{j=0}^{n-1} b_j 2^j = 2^n - |N|$$

cioè, la parte numerica dà per un numero negativo N il complemento a 2^n del valore assoluto di N.

Prima di illustrare come si effettuano le operazioni aritmetiche vogliamo analizzare come si ottiene il complemento a 2 di un intero positivo N rappresentato in binario come $b_{n-1}\, b_{n-2} \ldots b_0$. Dobbiamo a tale scopo, effettuare la sottrazione espressa da 2^n-N e ottenere la rappresentazione binaria $b'_{n-1}\, b'_{n-2} \ldots b'_0$ di (2^n-N). Si noti ora che

$$2^n-N=(2^n-1)-N+1= \sum_{j=0}^{n-1} 2^j - \sum_{j=0}^{n-1} b_j 2^j + 1 =$$

$$= \sum_{j=0}^{n-1} 2^j (1-b_j) + 1 .$$

Poichè $(1-b_j)$ è il "complemento" della cifra b_j (definiamo 1 come complemento di 0 e viceversa; in realtà in questo caso "complemento" sta per "complemento ad 1") possiamo ottenere le cifre b'_j effettuando successivamente le due operazioni seguenti:

1) Complementare ognuna delle cifre $b_{n-1}\ldots b_0$.
2) Aggiungere 000...01 a $(1-b_{n-1})(1-b_{n-2})\ldots(1-b_1)(1-b_0)$.

Supponiamo adesso che b_k sia la prima cifra uguale ad 1 a partire da destra, cioè, $b_{k-1}b_{k-2}\ldots b_0=00\ldots0$. Allora il complemento cifra per cifra di $b_k b_{k-1}\ldots b_0=100\ldots0$ è $011\ldots1$, per cui aggiungendo $000\ldots01$ otterremo un riporto che si propaga fino alla posizione 2^k dando luogo alla somma

$$100\ldots0 .$$

Questo risultato combinato col fatto che b_n deve essere posto uguale ad 1 dà la seguente:

O numero il cui complemento è uguale a se stesso

> <u>Regola per il complemento a 2</u>. Complementare tutte e sole le cifre a sinistra della cifra "1" meno significativa.(1)

ESEMPIO: Sia N=010110100. Allora la regola produce:

```
                 ┌──────────── 1 meno significativo
           N  =  0|10110100
-2^n+(2^n-N)  =  1|01001100
                 └──────────── posizione di segno
```

Consideriamo ora l'esecuzione delle operazioni di addizione e sottrazione. Consideriamo dapprima l'addizione di due operandi A e B di egual segno. Il primo requisito è che le n posizioni di cui si dispone siano sufficienti per rappresentare la somma; se A e B sono non-negativi, allora deve essere $(A+B) < 2^n$, mentre se A e B sono entrambi negativi allora deve essere $A+B \geq -2^n$. Se queste condizioni non sono verificate, diremo che si ha un "<u>trabocco</u>" o "<u>overflow</u>". Il riconoscimento di un trabocco è molto agevolato dalle considerazioni seguenti (supponiamo che $A \equiv a_n a_{n-1} \cdots a_0$, $B \equiv b_n b_{n-1} \cdots b_0$, siano i due operandi e che $S \equiv s_n s_{n-1} \cdots s_0$ sia la loro somma):

1) $A \geq 0$, $B \geq 0$. In questo caso $a_n = b_n = 0$. Se $(A+B) \geq 2^n$, nel qual caso si ha trabocco, si ha un riporto $c_n = 1$ alla posizione -2^n, che dà come risultato $s_n = 1$ e un riporto $c_{n+1} = 0$ dalla posizione -2^n. Cioè, riassumendo: $c_n = 1$, $c_{n+1} = 0$, $s_n = 1$.

ESEMPIO: Sia n=6

```
A=38 ≡ 0100110     B=45 ≡ 0101101
overflow          [01]011000   riporti
            38+    0100110+
            45     0101101
erroneo    -45     1010011
```

2) $A < 0$, $B < 0$. In questo caso $a_n = b_n = 1$. Considerando le sole parti numeriche, noi stiamo in effetti calcolando la somma S' di $(2^n - |A|)$ e $(2^n - |B|)$, cioè,

$$S' = (2^n - |A|) + (2^n - |B|) = 2^n + 2^n - (|A| + |B|) \;;$$

se $(|A|+|B|) > 2^n$, nel qual caso si ha trabocco, $S < 2^{n+1} - 2^n = 2^n$, per cui $c_n = 0$. Ma, poichè $a_n + b_n = 1+1 = 2$ c'è un riporto dalla posizione -2^n, cioè, $c_{n+1} = 1$. Riassumendo si ha: $c_n = 0$, $c_{n+1} = 1$, $s_n = 0$

(1) Si noti che $100\ldots0 \equiv -2^n$ è l'unico numero il cui complemento a 2 non è definito.

ESEMPIO: n=6.

```
A= -38 ≡ 1011010  ,   B= -51  ≡ 1001101
      overflow        [10]110000  riporti
               -38+    1011010+
               -51     1001101
      erroneo   39     0100111
```

Chiaramente la sottrazione per operandi di segno diverso è equivalente alla addizione di operandi di segno uguale: infatti nella sottrazione, si dovrà complementare il sottraendo, cosicchè i segni diventeranno identici.

Consideriamo ora la sottrazione di operandi di egual segno che è equivalente alla addizione di operandi di segno diverso. Questi due casi possono essere discussi contemporaneamente, senza perdita di generalità, considerando l'<u>addizione</u> di $A \geq 0$ e $B < 0$ (si noti che, così facendo, eliminiamo completamente la necessità del sottrattore). Abbiamo $a_n = 0$ e $b_n = 1$, e la somma S' delle parti numeriche ha valore

$$S' = A + (2^n - |B|) = 2^n + (A - |B|) \;.$$

Distinguiamo due casi:

1) Se $(A - |B|) \geq 0$ allora $S' \geq 2^n$, cioè, c'è un riporto $c_n = 1$. Questo riporto ha significatività aritmetica 2^n, mentre la significatività di b_n è -2^n. Perciò $b_n + c_n = 2^n - 2^n = 0$, cioè il riporto e il bit del segno <u>si elidono</u>, mentre la parte numerica contiene il corretto risultato non negativo $A - |B|$. Notare che il riporto c_{n+1} definito come riporto dalla posizione -2^{+n} è in questo caso uguale ad 1. Riassumendo:

$$c_n = 1, \; c_{n+1} = 1, \; s_n = 0$$

ESEMPIO: n=6.

```
A=38 = 0100110      ,      B= -25 = 1100111
nessun trabocco      [11]001100     riporti
                 38+   0100110+
                -25    1100111
      corretto  13     0001101
```

2) Se $(A - |B|) < 0$, allora $S' < 2^n$; cioè, $c_n = 0$, il bit del segno della somma è $s_n = b_n = 1$ e non si ha riporto c_{n+1}. Il risultato è perciò correttamente espresso come negativo e contiene nella parte numerica il

$2^n - (|B| - A)$

complemento a 2^n di ($|B|$ - A). Riassumendo:

$$c_n=0,\ c_{n+1}=0,\ s_n=1$$

ESEMPIO: n=6.

A=38 = 0100110 B= -45 = 1010011

nessun trabocco	00001100	riporti
38+	0100110+	
-45	1010011	
corretto	-7	1111001

L'esame delle condizioni di trabocco(che corrisponde ad una situazione di errore) e delle condizioni in cui la operazione è effettuata correttamente (non ci può essere trabocco se i due operandi da addizionare hanno segno diverso) conduce a formulare la seguente regola:

> "Si ottiene risultato corretto quando c_n e c_{n+1} sono uguali. Si ha trabocco ogni volta che c_n e c_{n+1} sono diversi".

*

3.2 - La rappresentazione in complemento a 1

Una variante della complementazione a 2 è la cosiddetta complementazione a 1, che è impiegata in alcuni calcolatori. La rappresentazione $b_n b_{n-1}\cdots b_0$ di un numero N in complemento a 1 fornisce il valore di N

$$N= -b_n(2^n-1)+\sum_{j=0}^{n-1} b_j 2^j \qquad (6)$$

cioè, il peso della posizione b_n è ora $-(2^n-1)$. Il massimo numero rappresentabile è sempre $011\ldots1 = (2^n-1)$ mentre il minimo è $100\ldots0=-(2^n-1)$ L'intervallo dei numeri rappresentabili è $-(2^n-1) \le N \le (2^n-1)$, cioè è simmetrico rispetto allo 0.

Si noti che questo intervallo contiene $(2^{n+1}-1)$ numeri, mentre vi sono 2^{n+1} configurazioni di (n+1) bit. Ciò significa che vi è un eccesso di una configurazione rispetto ai numeri da rappresentare, cioè che un numero ha una doppia rappresentazione. Infatti questo numero è N=0, che è rappresentato sia da 00...0 che da 11...1, cioè sia come numero positivo che come numero negativo: il valore della prima rappresentazione è ovvio; il valore della seconda è ottenuto applicando la (6),

ossia

$$N = -(2^n-1)+\sum_{j=0}^{n-1} 2^j = -(2^n-1)+(2^n-1)=0\ ,$$

come si voleva dimostrare.

Sia N un numero il cui bit di segno è 0 (cioè $b_n=0$, $N\ge0$) e sia $0b_{n-1}b_{n-2}\cdots b_0$ la sua rappresentazione. Vogliamo ottenere la rappresentazione $b'_n b'_{n-1}\cdots b'_0$ di -N in complemento a 1: chiaramente i coefficienti $b'_n,\ldots,b'_0$ sono incogniti. Poichè $-N \le 0$, il bit di segno b'_n è uguale a 1; la parte numerica $b'_{n-1}\cdots b'_0$ rappresenta il complemento a 2^n-1 di N, cioè rappresenta 2^n-1-N; formalmente

$$\sum_{i=0}^{n-1} b'_i\cdot 2^i = 2^n-1-N.$$

Poichè però, per la definizione del numero N, abbiamo

$$N = \sum_{i=0}^{n-1} b_i\cdot 2^i\ ,$$

sostituendo nella precedente uguaglianza si ha

$$\sum_{i=0}^{n-1} b'_i\cdot 2^i = 2^n-1-\sum_{i=0}^{n-1} b_i\cdot 2^i = \sum_{i=0}^{n-1} 2^i-\sum_{i=0}^{n-1} b_i\cdot 2^i = \sum_{i=0}^{n-1}(1-b_i)2^i$$

cioè $b'_i=(1-b_i)$ per i=0,1,...,n-1; ricordando inoltre $b'_n=1-b_n$, abbiamo la seguente semplice regola:

> Regola per la complementazione a 1: Complementare tutti i bit della rappresentazione del numero, compreso il bit di segno.

ESEMPIO: Dato N=0101101000, la regola dà

$$-2^n+1+(2^n-1-N)=1010010111$$

Per quanto riguarda l'esecuzione delle operazioni di addizione e sottrazione, iniziamo col considerare il caso di addizione di due interi di segno contrario (che è equivalente, come si è visto alla sottrazione di due interi del medesimo segno). Siano dati due numeri $A \equiv a_n a_{n-1}\cdots a_0$ e $B \equiv b_n b_{n-1}\cdots b_0$, e, senza perdere in generalità, siano $A \ge 0$ e $B \le 0$ (cioè $a_n=0$, A è non-negativo, e $b_n=1$, B è non-positivo).

La somma S' delle parti numeriche di A e di B è data da

$$S'=A+(2^n-1-|B|)=2^n-1+(A-|B|)\ .$$

Distinguiamo due casi:

1) Se $(A-|B|)\leq 0$, allora $S'\leq 2^n-1$, cioè non si ha riporto dalla posizione 2^{n-1} ($c_n=0$). In questo caso il bit di segno è $s_n=b_n=1$ (con $c_{n+1}=0$), cioè il risultato è correttamente espresso, poichè il bit di segno è pari a 1 e la parte numerica rappresenta $2^n-1-(|B|-A)$. Incidentalmente, si noti che quando $A-|B|=0$ si ha $S'=2^n-1$ e $s_n=1$, cioè il risultato A+B=0 viene ad essere rappresentato da 11...1.

ESEMPIO: Con n=7, si voglia addizionare 59 = .00111011 a -110 = = 10010001.

7 per la parte numerica e 1 per il segno

	c_{n+1} c_n	
	00 1100110	riporti
59+	00111011	
-110	10010001	
-51	11001100	

2) Se $(A-|B|)>0$, allora $S'>2^n-1$, cioè $S'\geq 2^n$. Si ha un riporto $c_n=1$, la cui significatività aritmetica è 2^n. Questo riporto si somma a $b_n=1$ e $a_n=0$, dando luogo a $s_n=0$ e $c_{n+1}=1$. Si noti, tuttavia, che il peso aritmetico di $b_n=1$ è -2^n+1, per cui $c_n 2^n+$ $+\ b_n(-2^n+1)=2^n-2^n+1=1$, ossia c_n e b_n non si elidono come nella complementazione a 2. Infatti in questo caso il segno della somma è corretto ($s_n=0$), ma la parte numerica rappresenta il risultato in difetto di 1; poichè questa situazione si verifica quando $c_n=1$ e $b_n=1$, il che implica $c_{n+1}=1$, si utilizza quest'ultimo evento per correggere il risultato. Questa correzione si esplica utilizzando il riporto c_{n+1} come un riporto nella posizione delle unità: è questo il motivo per cui la correzione descritta viene convenzionalmente denominata riporto ciclico (end-around carry). Questa è la più importante differenza tra l'esecuzione dell'addizione con operandi in complemento a 2 e quella con operandi in complemento a 1.

ESEMPIO: Si voglia addizionare 109 = 01101101 a -59 = 11000100.

	c_{n+1} c_n	
	11 0011000	riporti
109	01101101+	
-59	11000100	
49	00110001+	
	1	
50	00110010	

Consideriamo infine il caso di addizione di operandi aventi lo stesso segno.

3) $A\geq 0$, $B\geq 0$, cioè $a_n=0$, $b_n=0$. Si ha trabocco se e solo se $A+B\geq 2^n$, il che dà origine a $c_n=1$; si noti però che $c_n=1$, $a_n=0$, $b_n=0$ danno luogo a $c_{n+1}=0$, per cui ritroviamo la ben nota condizione $c_{n+1}=0$ $c_n=1$ come indicativa del trabocco.

4) $A\leq 0$, $B\leq 0$ cioè $a_n=1$, $b_n=1$. Si ha trabocco se e solo se $|A|+|B|\geq 2^n$. La parte numerica della somma ha il valore

$$S'=2^n-1-|A|+2^n-1-|B|=2^{n+1}-2-(|A|+|B|)$$

Poichè $|A|+|B|\geq 2^n$, si ha

$$S'\leq 2^{n+1}-2-2^n=2^n-2$$

cioè non si ha riporto alla posizione di segno, ovvero, $c_n=0$. Ma $a_n=b_n=1$ danno luogo ad un riporto $c_{n+1}=1$ e pertanto il trabocco è rivelato dalla condizione $(c_{n+1},c_n)=10$. Si noti che, in conformità con la regola del riporto ciclico, $c_{n+1}=1$ fà sì che 1 venga sommato alla parte numerica; poichè però $S'\leq 2^n-2$, abbiamo

$$S'+1\leq 2^n-1$$

cioè il riporto ciclico non può in nessun caso generare $c_n=1$.

Ciò conclude l'esame della rappresentazione in complemento a 1. E' opportuno notare che la complementazione a 2, mentre evita la complicazione del riporto ciclico, è però caratterizzata da maggiori complicazioni nell'operazione di complementazione vera e propria. Pertanto, nes-

suno dei due tipi di complementazione è chiaramente superiore all'altro per quanto concerne l'esecuzione dell'addizione di numeri relativi.

4. Rappresentazione in binario dei numeri decimali

Mentre sino ad ora ci siamo occupati della rappresentazione diretta dei numeri in binario, in questo paragrafo consideriamo una rappresentazione binaria indiretta, cioè la rappresentazione in binario delle cifre decimali e la relativa aritmetica.

Innanzitutto, poichè si hanno 10 distinte cifre decimali da rappresentare, occorrono almeno 4 cifre binarie per la loro codifica. E' però chiaro che se si scelgono 4 cifre binarie, delle $2^4=16$ configurazioni disponibili, solo 10 verranno utilizzate. Descriveremo due rappresentazioni di questo tipo, il BCD (binary coded decimal, decimale codificato in binario) e l'XS-3 (excess-three, eccesso-tre), che sono state impiegate in alcuni calcolatori per applicazioni gestionali, nonostante la tendenza presente sia verso l'adozione di rappresentazioni binarie pure.

1. BCD. Ciascuna cifra decimale d viene rappresentata da una quadrupla binaria $b_3b_2b_1b_0$ secondo la corrispondenza

$$d = \sum_{i=0}^{3} b_i 2^i .$$

La tabella che descrive il codice è data di seguito:

cifra decimale	rappresentazione BCD $b_3b_2b_1b_0$
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Si noti che le configurazioni binarie 1010, 1011, 1100, 1101, 1110, 1111 non sono utlizzate dal codice in questione. Ad esempio l'intero 538 ha la rappresentazione:

5 3 8
0101 0011 1000 = 010100111000.

I numeri negativi possono essere rappresentati in modulo e segno o in notazione complementata. Naturalmente, poichè la base effettiva è 10, si tratterà di complementazioni a 10 o a 9 (corrispondenti, rispettivamente, alle complementazioni a 2 e a 1 nel caso della base 2). Per quanto riguarda le operazioni aritmetiche, ci limitiamo a considerare l'addizione di due numeri positivi.
Siano i due addendi $U \equiv u_n u_{n-1} \cdots u_0$ e $V \equiv v_n v_{n-1} \cdots v_0$, dove sia le "u" che le "v" sono cifre decimali. Ad esempio

	11	Riporti
U	143	Addendo
V	375	Addendo
	528	Somma

E' chiaro che, poichè si hanno due addendi, i riporti possono assumere solo i valori 0 o 1. Se in ciascuna posizione la somma delle due cifre e del riporto non eccede 9, l'addizione avviene come se i numeri fossero in binario. Nel caso contrario, occorre che le apparecchiature producano la somma corretta ed un riporto alla cifra decimale successiva.

ESEMPIO:

4	0100	addendo
7	0111	addendo
11	1011	somma binaria risolta in:
1,1	1 0001	somma decimale + riporto

riporto ↑ cifra di somma

Concludiamo con un esempio di addizione.

ESEMPIO: Si vogliamo addizionare 386 e 451 rappresentati in BCD:

	1	0		RIPORTI DECIMALI
386	0011	1000	0110 +	ADDENDO
451	0100	0101	0001	ADDENDO
837	1000	0011	0111	SOMMA

Il codice BCD è detto un codice pesato, in quanto ciascuna posizione ha una significatività, o peso, fissi.

. XS-3. In questo codice - di tipo non pesato, come il lettore si renderà conto - ciascuna cifra decimale d è codificata dalla rappresentazione binaria di (d+3) (da cui il nome del codice). In altre parole se $d \equiv b_3b_2b_1b_0$, si ha

$$d+3 = \sum_{i=0}^{3} b_i 2^i .$$

La tabella del codice è data di seguito:

cifra decimale	XS-3 $b_3b_2b_1b_0$
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

L'apparente complicatezza di questo codice non è una bizzarria, in quanto si ottengono notevoli vantaggi nell'esecuzione delle operazioni aritmetiche. Ci limitiamo a considerare la somma di due cifre decimali positive, $d \equiv a_3a_2a_1a_0$ e $f \equiv b_3b_2b_1b_0$. Se eseguiamo la somma delle due rappresentazioni come se fossero numeri binari, il valore della somma sarà (d+3)+(f+3)=d+f+6. Supponiamo che le cifre in questione, d e f, appartengano alla posizione decimale delle unità. Un riporto prodotto nella loro addizione proviene dalla quarta posizione binaria ed ha peso 2^4=16, mentre il riporto dalla posizione decimale delle unità a quella delle decine deve valere 10: pertanto, un eventuale riporto ha un valore che eccede di 6 il valore corretto, per cui una correzione si rende necessaria. Distinguiamo allora due casi:

1) d+f+6 $\geq$ 16. In questo caso le apparecchiature di somma generano un riporto alla cifra decimale successiva, mentre le cifre binarie della somma rappresentano d+f+6-16=d+f-10. Poichè d+f+6 $\geq$ 16 è equivalente a d+f $\geq$ 10, il riporto è legittimo mentre le cifre binarie della somma, che rappresentano d+f-10, dànno nel codice XS-3

un risultato in difetto di 3 unità. Occorre quindi sommare una correzione pari a +3 a tutte le posizioni decimali che hanno originato un riporto.

2) d+f+6 < 16. In questo caso le apparecchiature di somma non generano alcun riporto alla cifra decimale successiva e le cifre binarie della somma rappresentano d+f+6, cioè un valore in eccesso di 3 unità rispetto al valore corretto, d+f+3, nel codice XS-3. Occorre quindi sommare una correzione pari a -3 a tutte le posizioni decimali che non hanno generato riporto. Questa correzione può essere effettuata sommando 13, cioè 2^4-3 (il complemento a 16 di 3) ed ignorando il riporto alla cifra decimale successiva.

ESEMPIO: Vogliamo addizionare 376 e 451 rappresentati in XS-3.

	0 ↖ 01111	1 ↖ 10000	0 ↖ 00000	RIPORTI DECIMALI / RIPORTI BINARI
376	0110	1010	1001+	ADDENDO
451	0111	1000	0100	ADDENDO
	1110	0010	1101+	RISULTATO PRIMA DELLA CORREZIONE
	1101	0011	1101	CORREZIONE
827	1011	0101	1010	SOMMA

Il vantaggio principale di questa notazione è che le apparecchiature per l'addizione non differiscono sostanzialmente da quelle che operano su numeri binari (a differenza del codice BCD); un inconveniente, invece, è la necessità della correzione. Un altro vantaggio, che citiamo semplicemente, è che il codice XS-3 è autocomplementante, cioè, il complemento binario a 1 della rappresentazione di una cifra decimale d rappresenta il complemento a 9 di d; ed esempio, 4 ≡ 0111 e il complemento a 1 di 0111 è 1000 ≡ 5, cioè il complemento a 9 di 4.

5. L'operazione di moltiplicazione di interi rappresentati in binario

Senza perdere in generalità possiamo limitare le nostre considerazioni ad operandi interi. Infatti, poichè tutti i numeri rappresentati nei calcolatori hanno un numero finito di cifre, qualunque numero razio-

nale rappresentato in un calcolatore è del tipo $2^{-k}.N$, dove N e k sono interi. Pertanto il prodotto di due numeri razionali $N_1.2^{-k1}$ e $N_2.2^{-k2}$, cioè $N_1N_2\,2^{-(k_1+k_2)}$, può essere ottenuto calcolando il prodotto N_1N_2 dei due interi N_1 ed N_2 e collocando la virgola binaria (k_1+k_2) posizioni a sinistra dell'estremità meno significativa della rappresentazione di N_1N_2. Analoghe considerazioni possono essere fatte per la divisione, che però non viene trattata in queste note.

Diversi algoritmi sono stati proposti per calcolare la rappresentazione del prodotto di due interi. Ci limitiamo a considerarne uno, dopo aver sviluppato un'adeguata motivazione. Riconsideriamo criticamente il metodo familiare per eseguire la moltiplicazione di due interi con carta e matita, ad esempio 158 x 243:

```
  158      Moltiplicando
  243      Moltiplicatore
  474
 732       Prodotti parziali
316
39394      Prodotto
```

I due operandi vengono chiamati moltiplicando M e moltiplicatore m, con funzioni ben diverse. Designando $m_s m_{s-1} \cdots m_0$ la rappresentazione decimale del moltiplicatore (nel nostro caso $m_2=2$, $m_1=4$, $m_0=3$), per ciascuna cifra m_i si calcola il "prodotto parziale" $m_i.M$ e lo si registra sul foglio, con l'avvertenza di scalarne l'incolonnamento una posizione a sinistra rispetto a $m_{i-1}.M$: in tal modo si esprime il fatto che m_i ha un peso 10 volte maggiore di quello di m_{i-1}. Quando si sono esaminate le cifre del moltiplicatore si sommano i prodotti parziali, opportunamente scalati, e si ottiene il risultato.

E' semplice rendersi conto che il prodotto può essere equivalentemente ottenuto accumulando i prodotti parziali, col loro corretto incolonnamento, via via che li si ottiene. In tal modo, l'esempio precedente può essere riformulato come segue:

```
  158      Moltiplicando
  243      Moltiplicatore
  474      1° prodotto parziale = 1^ somma parziale
 732       2° prodotto parziale
7794       2^ somma parziale
316        3° prodotto parziale
39394      Prodotto
```


Pertanto il ben noto metodo di moltiplicazione degli interi può essere formalizzato con il seguente algoritmo:

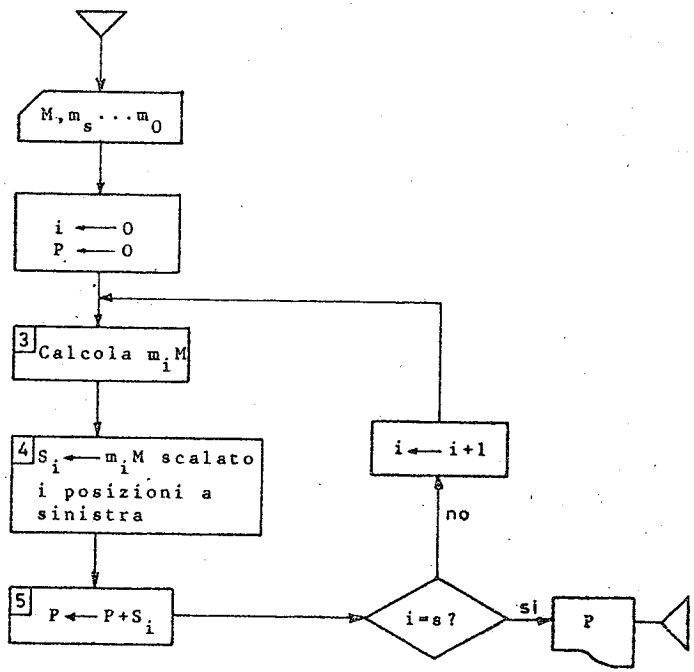


Figura 3.- Algoritmo elementare di moltiplicazione.

E' interessante notare che in nessun passo dell'algoritmo che abbiamo presentato si fa riferimento esplicito alla base di rappresentazione: infatti questo algoritmo è valido per qualunque base. Ovviamente le operazioni dei passi 3 e 5 devono essere eseguite nell'aritmetica della base adottata. Specificamente, nel caso della base decimale il calcolo del prodotto parziale (passo 3) è governato dalla ben nota "tavola pitagorica", mentre l'accumulazione del prodotto parziale (passo 5) è governata dalle regole della somma decimale. E' interessante notare che il passo 4 prescrive, attraverso l'operazione di "scalatura"

la moltiplicazione di $m_i M$ per r^i, dove r è la base scelta.

Nel caso che a noi interessa, cioè il caso di base 2 (aritmetica binaria), il passo 4 diviene particolarmente semplice, poichè m_i può assumere solo i valori 0,1. Quindi

$$m_i M = \begin{cases} M & \text{se } m_i=1 \\ 0 & \text{se } m_i=0. \end{cases}$$

Il passo 5 viene realizzato secondo la procedura descritta nel paragrafo 3 di queste note. Si noti infine che il calcolo del prodotto attraverso l'accumulazione dei prodotti parziali è particolarmente conveniente per gli elaboratori elettronici, in quanto questi ultimi sono generalmente progettati per l'addizione di due operandi alla volta. Siamo ora in grado di illustrare la moltiplicazione di due interi non-negativi rappresentati in binario.

ESEMPIO: Si abbiano $158_{10} = 10011110_2$ (moltiplicando)
e $27_{10} = 11011$ (moltiplicatore)

10011110	M=158
11011	m=27
10011110	1° prodotto (e somma) parziale m_0M
10011110	2° prodotto parziale $2m_1M$
111011010	2^ somma parziale
000000000	3° prodotto parziale $4m_2M$
111011010	3^ somma parziale
10011110	4° prodotto parziale $8m_3M$
11011001010	4^ somma parziale
10011110	5° prodotto parziale $16m_4M$
1000010101010	Prodotto = 4266

E' di estrema importanza osservare che se i due operandi sono espressi con n cifre binarie ciascuno, il prodotto può richiedere 2n cifre binarie per essere rappresentato. Infatti, il massimo valore del prodotto si ottiene quando ciascun operando assume il valore massimo, cioè (2^n-1). Pertanto

$$(2^n-1)(2^n-1)=2^{2n}-2^{n+1}+1,$$

numero la cui rappresentazione richiede 2n cifre binarie per $n\geq 2$.

Non vi è nulla di essenziale da aggiungere per quanto riguarda la moltiplicazione di interi non-negativi. Cosa accade nel caso di interi relativi? Chiaramente, non si ha alcuna complicazione nella rappresen-

tazione in modulo e segno, poichè il modulo del prodotto è un prodotto di interi non-negativi e il calcolo del bit segno del prodotto è governato dalla seguente tabella,

(m) \ (M)	0	1
0	0	1
1	1	0

Tabella III

dove (M) ed (m) sono i bit di segno rispettivamente del moltiplicando e del moltiplicatore.

Quando i numeri relativi sono rappresentati in notazione complementata si hanno due alternative:

1) Convertire gli operandi dalla rappresentazione complementata a quella in modulo e segno ed eseguire la moltiplicazione secondo le modalità prima descritte. Il segno del prodotto è calcolato secondo quanto specificato nella tabella III, e ovviamente è necessario complementare il prodotto quando il suo segno risulti negativo.

2) Sviluppare un algoritmo per eseguire la moltiplicazione usando direttamente gli operandi in notazione complementata.

Poichè la prima alternativa non presenta problemi nuovi, discuteremo la seconda, supponendo che gli operandi siano in notazione complementata a 2. Inoltre non ci soffermeremo ulteriormente sul calcolo del segno del prodotto e supporremo sia il moltiplicando che il moltiplicatore troncati del bit di segno (cioè limitati alle loro parti numeriche di n bit). Il caso in cui sia M che m siano non-negativi è già stato trattato. Rimangono allora tre casi da discutere:

(i) $M<0$, $m\geq 0$. In questo caso la parte numerica di M rappresenta $(2^n-|M|)$, mentre quella di m rappresenta m. Il prodotto delle parti numeriche ha quindi il valore

$$(2^n-|M|)m=m2^n-m|M|.$$

Poichè la parte numerica del prodotto deve rappresentare $2^{2n}-m|M|$ e poichè

$$2^{2n}-m|M| = (2^{2n}-m2^n)+(m2^n-m|M|)$$

è necessario sommare al risultato precedente la correzione $2^{2n}-m2^n = 2^n(2^n - m)$, cioè il complemento a 2^n di m scalato n posizioni verso sinistra.

ESEMPIO: Sia n=4, e siano M=-9 = (1)0111 e m=5 = (0)0101 (il bit di segno è dato tra parentesi)

```
  0111      2^4 - |M|        = 16-9
  0101          m            = 5
 -----
  0111
 0000
  0111
 -----
 0111
100011      m2^4 - m|M|  (prodotto parti numeriche)
1011        2^4(2^4-m)   Correzione
--------
11010011    2^8 - m|M|   =256-45
```

(ii) M≥0, m<0. In questo caso la parte numerica di M rappresenta M, mentre quella di m rappresenta $(2^n-|m|)$. Il prodotto delle parti numeriche ha il valore

$$M(2^n-|m|)=M2^n-M|m|.$$

Svolgendo un ragionamento analogo a quello del caso (i), si deduce la necessità di sommare una correzione $2^n(2^n - M)$ al risultato precedente.

(iii) M<0, m<0. In questo caso la parte numerica di M vale $(2^n-|M|)$ e quella di m vale $(2^n-|m|)$. Pertanto il prodotto delle parti numeriche ha il valore

$$(2^n-|M|)(2^n-|m|)=2^{2n}-2^n|M| - 2^n|m| + |M|\cdot|n| .$$

Poichè la parte numerica del prodotto deve rappresentare $|M|\cdot|m|$, occorre sommare la correzione $-2^{2n}+2^n|M| +2^n|m|$. Supponiamo di procedere come se le situazioni M<0 e m<0 si verificassero separatamente. Allora, per quanto discusso nel caso (i), M<0 richiede come correzione l'addizione del complemento a 2^n della parte numerica del moltiplicatore (che in questo caso è $2^n-|m|$) scalata n posizioni verso sinistra, cioè

$$2^n\left[2^n-(2^n-|m|)\right] = 2^n|m|.$$

Similmente, per quanto discusso nel caso (ii), m<0 comporta la correzione $2^n\left[2^n-(2^n-|M|)\right] = 2^n|M|$. Sommando queste correzioni al prodotto delle parti numeriche, otteniamo

$$2^{2n}-2^n|M| -2^n|m| +|M||m| +2^n|M| +2^n|m| = 2^{2n}+ |M|\cdot|m| .$$

Si noti però che, poichè $|M||m|\geq 0$, ne segue $2^{2n}+|M||m|>2^{2n}$, per cui si ha

il riporto (che viene ignorato) dalla posizione di peso 2^{2n}, e la parte numerica del risultato corretto vale esattamente $|M||m|$, come si desiderava.

ESEMPIO. Sia n=4, M=-9 = (1)0111 e m=-5 = (1)1011

```
  0111      2^4 - |M|
  1011      2^4 - |m|
 -----
  0111
 0111
10101
 0000
 10101
 0111
 1001101      prodotto parti numeriche
 1001         2^4|M|   correzione
--------
11011101
 0101         2^4|m|   correzione
00101101      |M|·|m|
```

ignorato → 1

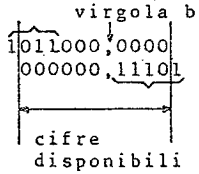
Possiamo così riassumere la precedente discussione nella seguente regola.

> "Per eseguire la moltiplicazione di numeri rappresentati in complemento a 2, occorre calcolare separatamente il bit di segno, e sommare al prodotto delle parti numeriche la correzione $2^n(2^n-m^*)$ se M<0 e $2^n(2^n-M^*)$ se m<0, dove M^* e m^* sono rispettivamente le parti numeriche del moltiplicando e del moltiplicatore."

6. Aritmetica in virgola fissa e aritmetica in virgola mobile

Le operazioni aritmetiche descritte nei precedenti paragrafi 3 e 5 (addizione e moltiplicazione) si riferivano ad operandi interi, ovvero, equivalentemente, ad operandi con parte frazionaria la cui virgola binaria aveva una collocazione fissa e comune a tutti i numeri (ad esempio, tra il terzo e il quarto bit da destra). In altre parole, nell'addizionare due numeri binari di n cifre nel modo descritto, era implicita l'assunzione che bit dello stesso peso fossero incolonnati, ossia che le virgole binarie di entrambi fossero pur esse incolonnate.

La scelta di una collocazione fissa per la virgola binaria (virgola fissa, fixed point) presenta però l'inconveniente che, quando si dispone di un numero relativamente piccolo di cifre (come sempre negli elaboratori elettronici), non sussiste la possibilità di rappresentare contemporaneamente numeri molto piccoli e numeri molto grandi. Ad esempio se si hanno 10 cifre binarie e si colloca la virgola binaria tra il quarto e il quinto bit da destra (vedi illustrazione), chiaramente non possiamo rappresentare i due numeri



88,0 e 0,90625 contemporaneamente. E' naturale asserire che in entrambi i casi vi è una "parte interessante" della rappresentazione, che si è indicata entro una graffa; questa parte interessante, tecnicamente chiamata "porzione significativa", è quella compresa, (per numeri in modulo e segno, come quelli della precedente illustrazione) tra la prima e l'ultima cifra non-zero della rappresentazione. E' desiderabile in molte applicazioni che la cifra di peso più elevato della porzione significativa di un numero abbia una collocazione fissa nell'ambito delle cifre disponibili: questa scelta porta naturalmente a sacrificare la collocazione fissa della virgola binaria. La notazione che ne scaturisce viene, per ovvi motivi, denominata in virgola mobile (floating point).(1)

La notazione binaria in virgola mobile è così definita. Ciascun numero N è rappresentato da una coppia di numeri (M,E). Il numero M, chiamato mantissa o talvolta coefficiente, ha n bit ed è in generale un numero razionale, comunemente rappresentato in complemento a 2. Il numero E è chiamato esponente (talvolta caratteristica), ha n_2 bit ed è intero, comunemente rappresentato in modulo e segno. La corrispondenza tra N,M ed E è stabilita dalla relazione

$$N = M.2^E \qquad (7)$$

Chiaramente, dato N, le coppie di numeri M ed E che soddisfano la (7) sono infinite. Ad esempio, N=14,5 può essere rappresentato dalle seguenti coppie (mantissa, esponente espressi in decimale):

(14,5 , 0) (7,25 , 1) (3,625 , 2) (1,8125 , 3) (0,90625 , 4) ecc.

Tuttavia non appena si fissi per la mantissa un intervallo compreso tra due potenze successive di 2, la determinazione della mantissa è univoca, da cui segue l'univocità della determinazione dell'esponente. Comunemente, per mantisse nella notazione in complemento a 2, si scelgono per M gli intervalli semiaperti $[\frac{1}{2},1)$ e $[-1,-\frac{1}{2})$, a seconda che M sia rispettivamente nonnegativa o negativa. Si fa un'eccezione per lo 0, che è rappresentato da M=0 e dal minimo esponente rappresentabile. I numeri in virgola mobile la cui mantissa appartiene a uno di questi due intervalli (o ad altra coppia equivalente di intervalli) vengono detti in forma normale(1)(2). Nel nostro caso la virgola è immediatamente a destra del bit di segno.

ESEMPIO. Siano n_1=10, n_2=5. I bit di segno sono indicati tra parentesi. Si studino i seguenti casi ($N=M.2^E$):

N	M	E
3392	(0)110101000	(0)1100
-412	(1)001100100	(0)1001
0,078125	(0)101000000	(1)0011
-0,046875	(1)010000000	(1)0100

L'aritmetica in virgola mobile, cioè l'esecuzione delle operazioni elementari con operandi rappresentati in virgola mobile, consiste ovviamente di due componenti: l'aritmetica delle mantisse e l'aritmetica degli esponenti. Consideriamo ora l'esecuzione delle operazioni

(1)- E' opportuno osservare che mentre nella notazione binaria in modulo e segno la porzione significativa di un numero inizia con l'1 di peso più elevato, nella notazione complementata (ad esempio a 2) ciò vale solo per i numeri non-negativi. Invece per i numeri negativi la porzione significativa inizia con lo 0 di peso più elevato. Il lettore può giustificare questa asserzione ricordando la regola di complementazione.

(1)- La notazione in virgola mobile non è una peculiarità dell'aritmetica dei calcolatori. Si pensi all'espressione di molte costanti fisiche, come il numero d'Avogadro o la carica dell'elettrone, per le quali, comunemente si sceglie una mantissa nell'intervallo [1,10). L'intervallo dei numeri rappresentabili, a parità di numero di cifre impiegate, è enormemente espanso dalla notazione in virgola mobile.

(2)- Si ricordi che il complemento di 10...0 non è definito e si noti che quello di 010...0 non è in forma normale. A ciascuno di questi due numeri si assegnerà allora un complemento approssimato.

di addizione-sottrazione e di moltiplicazione.

1) Addizione-sottrazione. Come al solito, possiamo limitarci a considerare l'addizione di due numeri relativi (M_1,E_1) e (M_2,E_2). L'operazione di addizione è descritta dall'algoritmo di figura 4. Ovviamente, cifre di peso identico di M_1 ed M_2 sono correttamente incolonnate se e solo se $E_1=E_2$. Pertanto quando i due esponenti sono diversi, occorrerà scalare a destra la mantissa relativa all'esponente minore di un numero di cifre pari alla differenza ΔE degli esponenti, al fine di ottenerne il corretto incolonnamento con l'altra mantissa. A questo punto si può eseguire l'addizione delle mantisse, e l'esponente della somma

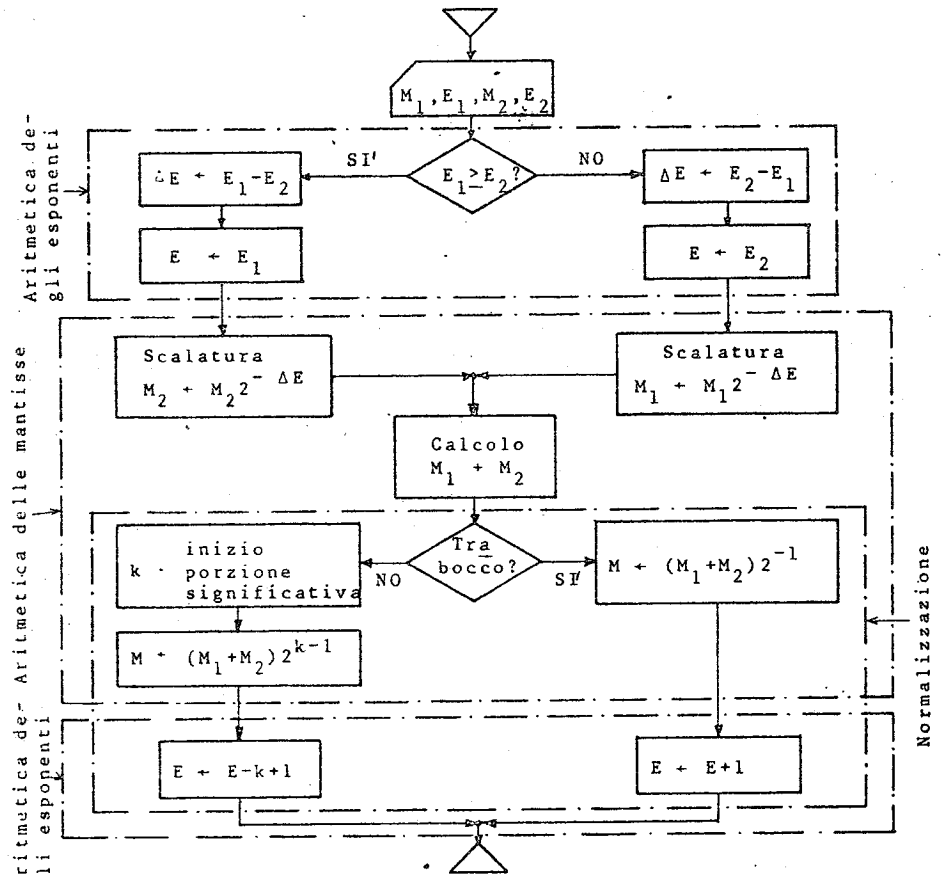


Fig 4 - Algoritmo di addizione in virgola mobile.

(M_1+M_2) è il maggiore dei due esponenti E_1 ed E_2. Se l'addizione dà trabocco, chiaramente M_1+M_2 appartiene, se positivo, all'intervallo $[1,2)$, se negativo all'intervallo $[-2,-1)$: in entrambi i casi la somma viene normalizzata scalandola a destra una posizione (cioè moltiplicandola per 2^{-1}) ed incrementando di 1 l'esponente. In assenza di trabocco, occorre determinare, a partire da sinistra, l'inizio della porzione significativa del risultato: se ciò avviene alla k-esima cifra a destra della virgola, la normalizzazione si ottiene scalando (M_1+M_2) di (k-1) posizioni verso sinistra e decrementando di (k-1) l'esponente. Nel primo caso di normalizzazione si approssima perdendo una cifra a destra; nel secondo, si approssima introducendo da destra zeri non significativi. Ciò conferma il carattere intrinsecamente "approssimato" dell'aritmetica in virgola mobile.

2) Moltiplicazione. A differenza dell'addizione-sottrazione che, come si è visto, offre considerevoli complicazioni, la moltiplicazione è molto semplice con operandi in virgola mobile. Infatti, in prima analisi, poichè

$$M_1 \cdot 2^{E_1} \times M_2 \cdot 2^{E_2} = M_1 M_2 \; 2^{E_1+E_2},$$

sembrerebbe sufficiente sommare gli esponenti e moltiplicare le mantisse (secondo i metodi illustrati nei precedenti paragrafi). Ciò è corretto ogniqualvolta M_1M_2 appartiene agli intervalli di normalizzazione. In generale, però, il prodotto M_1M_2 appartiene agli intervalli $(-1,-1/4)$ e $[1/4,1]$ per cui una normalizzazione può essere necessaria, incrementando o decrementando di 1 l'esponente (E_1+E_2) e scalando corrispondentemente il prodotto delle mantisse.

7. Note e bibliografia

L'aritmetica dei calcolatori è trattata in un numero vastissimo di libri di testo e di articoli in riviste specializzate. Senza voler far torto a nessuna delle pubblicazioni non citate, mi limito a menzionare i seguenti libri:

1) R.K.Richards, Arithmetic Operations in Digital Computers, D.Van Nostrand Co., Princeton, N.J., 1955.

Si tratta di un testo antiquato per quanto concerne la realizzazione di unità aritmetiche e di reti logiche, ma sempre valido per la sua chiara esposizione dell'aritmetica binaria.

1) Y.Chu, Digital Computer Design Fundamentals, McGraw-Hill, New York, 1962.

I capitoli 1 e 2 contengono una esposizione abbastanza completa, anche se un poco concisa, dei vari metodi impiegati nella realizzazione dell'aritmetica digitale.

3) Taylor L.Booth, Digital Networks and Computer Systems, J.Wiley, New York, 1971.

Un buon testo introduttivo agli elaboratori. Il Capitolo II contiene una discussione concisa dell'aritmetica.

4) H.W.Gschwind, Design of Digital Computers, Springer-Verlag, New York, 1967.

Contiene un buon capitolo (Capitolo 2) sulla rappresentazione dei numeri e molte informazioni sulla struttura delle unità aritmetiche (Sottocapitolo 8.1).

per il lettore particolarmente maturo, sia matematicamente sia nel campo dei calcolatori, consiglierei il seguente testo di carattere avanzato:

D.E.Knuth, The Art of Computer Programming, Vol.2, Addison-Wesley, Reading, Mass, 1971 (pagg.161-451).

Queste note costituiscono l'ampliamento di una piccola dispensa da me scritta in inglese alcuni anni fa e tradotta in italiano dal Dr.G. Pacini, che ringrazio per la collaborazione. Ringrazio inoltre lo stesso Dr.Pacini e il Prof.C.Montangero per alcuni suggerimenti volti a migliorare la qualità dell'esposizione.