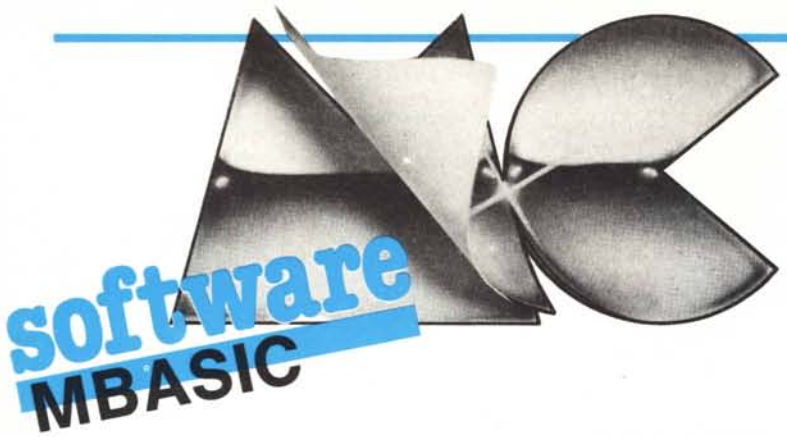

La gestione dei numeri

In questa puntata parleremo di un argomento alquanto delicato e forse leggermente complesso, seppure non di immediata utilizzazione: la gestione delle quantità numeriche da parte dell'MBASIC.

Dicevamo che ciò non è tutto sommato di immediata utilizzazione, in quanto il programmatore non si accorge minimamente di ciò che accade all'interno del suo computer quando si effettuano dei calcoli, ad esempio una semplice addizione.

Parliamo però di questo argomento per introdurre poi la prossima puntata un argomento ben più interessante ed utilizzabile e cioè le istruzioni USR e CALL, di solito "maltrattate" dai manuali dei personal computer.

Per quanto riguarda l'argomento di questa volta, parleremo in particolare di quella che è la "rappresentazione interna" delle quantità intere, reali, in doppia precisione: non ci occuperemo (almeno questa volta) della memorizzazione delle variabili e perciò del loro formato.

Iniziamo dunque la "carrellata".

I numeri interi

Come senz'altro sarà già noto i numeri interi vengono rappresentati con 2 byte, in notazione "complemento a 2" (vedasi ad esempio nel n. 36 di MC, nella rubrica "L'Assembler dello Z80").

In questo modo si possono rappresentare numeri positivi compresi tra 1 e 32767, nonché quelli negativi compresi tra -1 e -32768, più ovviamente 1 o 0.

In particolare, considerati i 2 byte come un insieme di 16 bit, in realtà se ne usano solo 15, lasciando al più significativo il compito di indicare se il numero è positivo (se il bit è 0) oppure negativo (se il bit è 1).

Il fatto che si utilizzino 15 bit spiega inoltre perché il massimo valore positivo è 32767 ("0" seguito da 15 "1") e non, come siamo abituati, 65535, che si ottiene viceversa con 16 bit pari ad 1.

Facciamo alcuni esempi di rappresentazioni di numeri interi dotati di segno, senza però soffermarci sul perché è stata scelta la rappresentazione "complemento a 2" per i numeri stessi.

numero intero	rappresentazione binaria	esadecimale
23	0000000000010111	0017
1000	0000001111101000	03E8
1	0000000000000001	0001
-23	1111111111101001	FFE9
-1000	1111110000011000	FC18
-1	1111111111111111	FFFF

Fin qui, come detto, non c'è niente di nuovo, a parte eventualmente il fatto che la rappresentazione è complementata.

Passiamo dunque ai numeri reali, per i quali la gestione è ben più complessa.

I numeri in singola precisione

Questo "tipo" di quantità numeriche ci consente, come sappiamo, di rappresentare valori ben più grandi di 32768, arrivando fino a quantità come 10^{38} e cioè che noi esprimiamo in "notazione esponenziale".

Con quest'ultimo termine sappiamo infatti che si possono rappresentare tutti i numeri, anche di ordini di grandezza differente.

Infatti numeri come 1000000000 e 0.005 possono essere scritti in un modo particolare, formato da una "mantissa" e da un "esponente".

Dato che 1000000000 può essere espresso come 10^9, ecco che la sua rappresentazione in notazione esponenziale sarà data da "1E9" dove "1" è la mantissa e "9" è l'"esponente" di 10.

Analogamente 0.005 può essere rappresentato in notazione esponenziale con "5E-3", dove ora l'esponente di 10 e "-3", in quanto sappiamo che vale l'uguaglianza $0.005 = 5 \times 10^{-3}$

Con questi esempi abbiamo più che altro voluto sottolineare il fatto che "noi" siamo abituati a fare i conti in "base 10" e perciò troviamo utile la notazione esponenziale.

Invece il nostro computer ragiona, come sanno anche i bambini, in "base 2" (ed in più "complementata") e perciò per poter rappresentare numeri più grandi di 32767, dovrà usare una "notazione esponenziale" però in "base 2", con alcune ottimizzazioni e regolette: anche se all'inizio ciò può sembrare molto macchinoso, a lungo andare ci si fa l'abitudine.

Dimenticavamo di dire che una quantità reale occupa uno spazio di memoria dato da 4 byte: il problema è di riuscire a rappresentare con $4 \times 8 = 32$ bit il maggior numero possibile di valori.

Per introdurre il concetto di "notazione esponenziale in base 2", conviene partire dall'esempio di due numeri aventi ordine di grandezza molto diversi, in modo da cercare una rappresentazione similare nei due casi e sulla quale il computer possa lavorare.

Supponiamo perciò di analizzare i due valori 50000 e 0.0625: innanzitutto dobbiamo scriverli in binario.

Per il primo si ha il valore esadecimale 0C350H, equivalente al binario

1100001101010000

Il secondo invece è un valore decimale che vale 2^{-4} e perciò in binario si esprime come

0.0001

Torniamo ora al primo valore: dato che si tratta di un valore intero, la sua virgola "binaria" (analoga alla virgola "decimale", ma qui stiamo lavorando in base 2...) si troverà subito a destra della cifra meno significativa.

Analogamente a quanto si fa nella notazione esponenziale supponiamo di "spostare" verso sinistra la virgola, ma così facendo dobbiamo moltiplicare il numero proprio di quanto lo abbiamo diviso: in

particolare, spostando la virgola di un posto verso sinistra si divide il valore per 2, cosicché avremo:

110000110101000.0 ♯ 2

Abbiamo messo il simbolo "♯", al posto di un "x", in quanto stiamo usando una simbologia non uniforme, formata da sequenze di bit da un lato e "numeri decimali" dall'altro: l'importante è capire ciò che stiamo eseguendo.

Ogni volta che ci spostiamo perciò di un bit a sinistra dovremo moltiplicare per due: alla fine avremo

$.1100001101010000 \sharp 2^{16}$

Nel secondo caso avremo, dopo analoghi passaggi

$.1 \sharp 2^{-3}$

Con un altro esempio ci accorgiamo pure di un'altra notevole caratteristica di questo tipo di notazione: consideriamo il valore 10, che in binario si esprime con 1010, il quale a sua volta si può trasformare in

$.1010 \sharp 2^{4}$

Riscriviamo dunque una tabellina sulla quale faremo le nostre considerazioni

valore decimale	notazione esponenziale mantissa	esponente
50000	.110000110101	16
0.0625	.1	-3
10	.101	4

dove dalla mantissa abbiamo eliminato gli "zeri" non significativi e dove l'esponente è espresso in decimale.

Analizzando la mantissa scopriamo che in ogni caso il primo bit dopo la virgola è un "1" (chi non fosse convinto può provare con altri esempi): allora nella rappresentazione all'interno del computer tale bit viene completamente eliminato (!) e sostituito dal bit di segno (0 = positivo, 1 = negativo).

Dato che abbiamo a disposizione 4 byte per rappresentare ogni numero, decidiamo di riservare il primo per l'esponente e i successivi 3 per la mantissa, cominciando a scrivere i bit "da sinistra" e cioè ponendo il bit di segno come MSB del byte più a sinistra.

Nei tre casi visti prima, le mantisse saranno

01000011 01010000 00000000
00000000 00000000 00000000
00100000 00000000 00000000

dove, ripetiamo, è stato sostituito il bit più significativo (cioè quello più vicino alla virgola) con il bit di segno, in tutti e tre i casi nullo: ora dobbiamo aggiustare il tutto con l'"esponente", che rispettivamente valeva 16, -3 e 4. Ora per convenzione si pone nel primo, dei 4 byte a disposizione, il "valore dell'esponente +128" e cioè rispettivamente 144, 125 e 132, ovviamente espressi in binario e cioè

10010000
01111101
10000100

Riassumiamo dunque i tre risultati sotto forma di 4 byte esadecimali:

il valore	diventa...
500000	90 43 50 00
0.0625	7D 00 00 00
10	84 20 00 00

Se qualcuno volesse per esercizio verificare quanto detto finora, diamo altri due valori già codificati

500.3	89 7A 26 66
0.01	7A 23 D7 0A

Da quanto abbiamo detto perciò abbiamo a disposizione per codificare la mantissa $7+2\times8=23$ bit, più 1 il segno ed altri 8 per l'esponente.

Possiamo a questo punto calcolare quale è il maggior numero in valore assoluto, rapprensentabile con questa notazione: avrà il più alto esponente possibile e cioè 127 (che sommato a 128 dà 255 e cioè il massimo valore che possiamo porre nel byte di esponente), avrà il segno positivo (e perciò il secondo byte comincerà per 0) e come mantissa avrà 23 "uni": espresso in esadecimale come i precedenti valori, avremo FF 7F FF FF, che corrisponde al valore decimale 1.70141E38.

Viceversa il più piccolo numero in valore assoluto rappresentabile è quello in cui solo l'ultimo bit, l'LSB, della mantissa è pari ad 1, con tutti gli altri bit nulli e con il byte di esponente pari ad 1, corrispondente ad un esponente del 2 pari a -127: avremo così la quaterna di byte data da 01 00 00 01, corrispondente al valore decimale 2.93874E-39.

I numeri in doppia precisione

Per quanto riguarda la rappresentazione dei numeri in doppia precisione, valgono le stesse considerazioni del paragrafo precedente, salvo che ora, invece di 4 byte (3 di mantissa + 1 di esponente) ne abbiamo 4 in più per la mantissa e cioè in tutto 8 (7 per la mantissa e 1 per l'esponente).

Questo fatto non comporta un ulteriore ingrandimento del range dei numeri rappresentabili dal computer, ma una precisione doppia: ora, al posto della lettera "E" di separazione tra la mantissa e l'esponente, nella notazione decimale verrà usata la lettera "D".

Lasciamo perciò all'abile lettore il compito di verificare che il valore decimale corrispondente al massimo valore assoluto rappresentabile con la doppia precisione, che a sua volta è espresso dai seguenti 8 byte esadecimali:

FF 7F FF FF FF FF FF FF

è dato da 1.701411834604692D+38. MC

PER IL TUO COMMODORE 64

EASY COMPUTING

Ora EASY COMPUTING ti dà una mano per far funzionare al meglio il tuo COMMODORE 64. Una organizzazione amica ed efficace famosa in Europa, e da oggi anche in Italia.

EASY COMPUTING ti offre la più vasta gamma di prodotti originali per il COMMODORE 64, tradotti in italiano, per un immediato utilizzo, sia nel campo professionale che nel tempo libero. Con il vantaggio di ricevere tutta la documentazione relativa al programma che ti interessa direttamente a casa tua. Basta compilare il coupon o scrivere direttamente a EASY COMPUTING - Via A. Bertani 24 - 50137 Firenze.

Questi i principali programmi che EASY COMPUTING ha selezionato per te:

SUPERSOFT - MUSIC MASTER, BUSICALC 2, BUSICALC 3, TOOLKIT, VICTREE, ZOOM, INTERDICTOR PILOT, MIKRO ASSEMBLER e una scelta di VIDEOGAMES intelligenti.

ABACUS - ZOOM PASCAL, SUPER DISK UTILITIES, SCREEN GRAPHICS, ULTRABASIC, SYNTHY 64, VIDEOBASIC, GRAPHICS DESIGNER, TAS, CADPAK, CHARTPAK.

VIZA - VIZASPELL, VIZAWRITE.

ANIROG - Per la prima volta in Italia decine di videogames originali, considerati come i più elaborati e affascinanti del mercato europeo.
OXFORD PASCAL, HARDCOPY.

HARDWARE - SUPERSKETCH, VIDEO GRAPHIC DIGITISER, LIGHT PEN, 4 SLOT MOTHERBOARD.
INTERFACCE: SERIELINK/RS, SERIELINK, CENTROSERIAL, PRINTLINK, etc.

EASY COMPUTING

VIA A.BERTANI 24 FIRENZE

Sono interessato a ricevere il catalogo generale EASY COMPUTING, gratuitamente e senza impegno, al seguente indirizzo:

Nome _____

Cognome _____

Indirizzo _____

Città _____ CAP _____

Professione _____

Tel. _____

MC