In order to use the TB access to the on-board calculator routines, some understanding of TB variable storage is necessary. Since TB is an integer basic with a maximum range of -32768 to +32767, variables are stored in one 16 bit (2 byte) location. The so-called "string" variable storage areas are assigned by TB to start immediately following the last line of your Basic program. Thus, @(0) occupies the first two bytes following your program, @(1) occupies the second and so on.

The Arcade on-board math routines expect to operate on n digit numbers where each decimal digit occupies the four low-order bits of consecutive bytes. For "ease" of interfacing TB (with its double-byte accesses) to the on-board math routines (which use single-byte accesses), the TB interface routines do some house-keeping as well as perform the ferrying of math requests back and forth between the running TB program and the on-board math routines. This house-keeping expects certain data formats as input and provides certain data formats as output.

First on the list of data formats is the requirement that all interfacing between TB and the math routines be via 18 consecutive variable storage locations for each argument (Augend, Addend, Sum; etc.). The simplest way to honor this requirement appears to be the use of the "string" variables (@(n)).

The second requirement is that each variable must be formatted in such a manner that the lower 4 bits of each variable contain the information necessary to define one digit of the argument. This isn't as bad as it may first appear, since the remaining 12 bits of each variable in an argument aren't used and may be anything. Thus one variable represents one digit in binary, BCD, ASCII (TV/KP) or whatever is handy.

Now comes the toughie! Each 18 variable argument is formatted rigidly. For example if one argument is given as @(0), then variables @(0), @(1). . . . .@(17) are expected to contain the argument, one digit per variable. The digits must be arranged such that @(15) contains the MSD, @(8) contains the LSD of the integer portion and @(7) contains the MSD, @(0) contains the LSD of the fractional portion. In this example @(17) contains the sign of the argument (48 for positive, 57 for negative). The variable @(16) should be 48 (it can be used, but care must be exercised since it may only be used for addition). The calculation result is returned in like manner, except @(17) may now contain a 63 indicating OVERFLOW, and all digits are returned in their ASCII form (printable by TV=).

SYNTAX: $signarg1,arg2,arg3

$ Required, signifying a calculator request

sign The operation to perform (+ - x or ÷)

arg1 The location of the LSD of the first input argument

arg2 The location of the LSD of the second input argument

arg3 The location of the LSD of the area in which to store the result

The positioning of the arguments follows standard algebraic logic. Thus,

$+arg1,arg2,arg3 implies arg1+arg2=arg3

$-arg1,arg2,arg3 implies arg1-arg2=arg3

$xarg1,arg2,arg3 implies arg1xarg2=arg3

$÷arg1,arg2,arg3 implies arg1÷arg2=arg3

EXAMPLES

$+@(0),@(18),@(0)

$-@(0),@(18),@(0)

$x@(0),@(18),@(0)

$÷@(0),@(18),@(0)

Tom Wood
3 Feb 79