Integer Addition

George E. Collins

September 8, 1998

The list representation of integers in SACLIB makes it a rather challenging task to write a program to efficiently add two arbitrary integers. ISUM is such a program. It consists of 12 steps and 223 lines. All SACLIB source programs can be found in /usa/saclib/saclib2.1/src.

Step 1 disposes of the trivial case in which either input, A or B, is zero. Step 2 then disposes of the case in which either input is single-precision. Step 3 determines the sign of each input. It does this using a program ISIGNF that computes the sign of any integer. Notice that to do so requires scanning the list until a nonzero β-digit is found. To simplify the task, Step 4 converts either input that is single-precision to a list of length 1. Step 5 branches according to whether the two inputs have the same or different signs. Step 6 deals with the case that both signs are positive, Step 7 with the case that both are negative. Whereas most programs obtain list cells using COMP, ISUM achieves greater efficiency by storing sum digits directly in the available cell list AVAIL (a global variable), then disconnecting the portion of AVAIL that was used when the sum is complete. If necessary, ISUM calls GC, the garbage collection program.

Steps 8 through 12 carry out the successive phases of the process that is required when the two inputs have opposite signs. Step 8 adds corresponding pairs of digits from the two inputs, obtaining digits that may be positive, negative or zero. It keeps track of the last nonzero sum digit that it has generated, if any. If none, and if the end of each input has been reached, then the sum is zero, and Step9 deals with this case. If Step 8 stopped when it reached the end of the shorter input, then Step 10 then continues to inspect digits of the longer input until the sign is determined. Step 11 then normalizes the sum, that is, it forces each digit to have the same sign as the sum, or else to be zero, through generation of carries. Step 12 deletes from the completed sum any leading zero digits. It also appends any high-order portion of the longer input that was not processed in the earlier steps.

IDIF computes the difference of two integers. It is nearly identical to ISUM. INEG computes the negative of any integer. It is, of course, very trivial. IABSF computes the absolute value of any integer; it merely applies ISIGNF, then, if appropriate, INEG.

ILOG2 computes the bit length of any integer A. This is the same as the floor function of $\log_2 |A|$, plus 1, with the exception that by convention the bit length of zero is defined to be zero. ILOG2 uses a subprogram, DLOG2, that computes the bit length of any β-digit. The bit length of an integer of length $n > 1$ is $(n-1)\zeta$ plus the bit length of its leading β-digit. By definition, $\zeta = \log_2 \beta$; this is a system parameter that is stored in the global variable ZETA. DLOG2 uses a subprogram PDLOG2 that computes the bit length of any positive β-digit. PDLOG2 uses successive comparisons with precomputed powers of 2 that are are stored in the global array TABP2, togther with a small number of final shifts to quickly obtain the answer.

Some other useful but simple integer functions are IMAX(A,B), IMIN(A,B) (maximum and minimum), IEVEN(A) and IODD(A) (predicates that are true or false), and ICOMP(A,B), the value of which is the sign of $A - B$. However ICOMP does not compute the difference; it obtains the answer more directly.