Integer Division

George E. Collins

September 18, 1998

Analagous to the program DPR that is needed for multiplication, SACLIB has an assembly language program DQR (digit quotient and remainder) that takes as input a two-β-digit dividend and a β-digit divisor, and produces as output two β-digits, a quotient and a remainder. DQR computes the quotient one bit at a time by subtraction and shifting. Its computing time on ren is approximately 0.94 microseconds.

The SACLIB program IQR divides an arbitrary integer A by an arbitrary nonzero integer B. obtaining a quotient Q and remainder R such that Q is the floor function of A/B and $R = A - BQ$. If B is a β-digit, IQR uses the special subprogram IDQR, which we will not discuss. Otherwise, $|A|$ and $|B|$ are copied into arrays using IFLTA and the method described in Section 4.3.1, Algorithm D, of [2] is used. Note that if Q' is the integral part of $|A|/|B|$ and $R' = |A| - |B|Q'$ then $Q = Q'$ if $\mathrm{sign}(A) = \mathrm{sign}(B)$ and $Q = -Q'$ otherwise. Also, $\mathrm{sign}(R) = \mathrm{sign}(A)$.

Step 4 of IQR detects cases where the quotient is obviously 0. Theorem A, page 271, of [2] shows that if the leading β-digit of the divisor is at least $\beta/2$, then dividing the two high-order digits of the divident by the leading digit of the divisor will produce a number that is greater than or equal to the leading digit of the quotient. Theorem B, immediately following, shows that if the leading β-digit is at least $\beta/2$, then the quotient digit obtained will be at most 2 larger than the correct quotient digit. Therefore Step 5 of IQR "normalizes" the divisor, multiplying it by the appropriate power of 2 to achieve this condition. Notice that here we have an application of PDLOG2, that the multiplication is done by shifting, and that the dividend must also be multiplied by the same power of 2. The quotient digit approximation determined by this two-digit by one-digit division is computed in the first 10 lines of Step 10 of IQR. The division is performed by PNDQR, not DQR. On a prior architecture PNDQR was faster than DQR, but on ren the times of the two programs are nearly the same. Next the method of Step D3 of Algorithm D of [2] is employed. This method adjusts this first aproximation to become the integer part of the three highest order β-digits of the dividend divided by the two highest β-digits of the divisor. The adjustment is made without division since Theorem B guarantees that the adjusted quotient is larger by at most two. Now we know that our approximate quotient digit is either correct or too large by only one. Exercise 21 at the end of Section 4.3.1 shows that it is too large with a probability of at most $2/\beta$.

Step 11 multiplies the divisor by the quotient digit approximation q and subtracts the product from the dividend, carrying out the multiplication and subtraction as a single process. If the result is negative, Step 12 adds the divisor back and decreases q by 1. After all quotient digits have been computed, Step 14 "denormalizes", dividing the remainder by the power of 2 by which we multiplied at the beginning, and imposing the correct signs on quotient and remainder. Also, the quotient and remainder are converted to lists.

Table 1 shows times T in microseconds for using IQR to divide a $2n$ β-digit integer by an n β-digit divisor.

For nonzero integers A and B of lengths m and n respectively, with $m \geq n$, the computing time of IQR is codominant with $n(m-n+1)$. If $m < n$, its computing time is codominant with n.

The program IEQ (integer exact quotient) can be used in lieu of IQR if it is known in advance, as it often is, that A is exactly divisible by B. The method used by IEQ is due to T. Jebelean, [1]. In this method the low-order digits of the quotient are computed first. Here we will describe the method for the case that the number system radix is a power of 2, as in SACLIB, but it can be used more generally. First we arrange for B to be odd, as follows. If B is divisible by 2^k, with k maximal, then A is also divisible by 2^k since $A|B$. So we divide both A and B by 2^k (by shifting). This does not change the quotient Q, and now B is odd. Let b_0 be the low order β-digit of B. b_0 is odd so it is relatively to β, and has an inverse, b_0' modulo β. Let a_0 be the low order digit of A and let $c_0 = a_0 b_0'$ modulo β. c_0 must be the low order digit of Q, and if we subtract $c_0 B$ from A, the result has 0 as low order digit, and is divisible by β. If we call this result A', we then have $A = A'\beta + c_0 B$. The next digit of Q can be obtained in the same way with A' in place of A. The method as described thus far has a computing time that is the same, to within codominance, as IQR. But it can be made faster if we observe the following. When we multiply B by c_0 and subtract from A, we need not, in general, carry out the process to completion as we proceed from low order digits to higher order digits. We only need to proceed through a_{m-n} because the quotient will have at most $m-n+1$ digits. Any higher order digits of A would never be used in the algorithm so they need not be computed. However, if $m-n+1 \geq n$ then no saving is achieved in computing c_0. Jebelean shows that the number of digit multiplications used by this method is

$$\begin{array}{ll} (m-n+1)(m-n+2)/2 & \text{if } m+1 \leq 2n, \\ mn - 3n(n-1)/2 & \text{otherwise.} \end{array}$$

Table 2 shows the times required by IEQ to divide a 2 $n\beta$-digit number by an n β-digit number. As expected, these times are approximately one half of the corresponding times for IQR.

Table 1: IQR Times

n	T
1	2.59
2	6.32
3	10.3
4	14.5
5	17.2
10	47.2
20	141
40	478

Table 2: IEQ Times

n	T
1	2.25
2	3.97
3	5.41
4	7.12
5	9.67
10	23.5
20	66.9
40	228.

References

[1] Tudor Jebelean. An Algorithm for Exact Division. J. Symbolic Computation, Vol. 15 (1993), pp.169-180.

[2] Donald E. Knuth. The Art of Computer Programming, Vol. 2, 3rd Ed. Addison-Wesley, 1997