Extended Precision Computation

Stephen J. Rogowski

How to compute to a practically unlimited number of places with just the memory your micro has on board.

No New Hardware

No black boxes are needed. No new boards need be installed. The ability to compute values beyond the normal precision of BASIC, FORTRAN—or any language for that matter is not a function of memory size or even the sophistication of the operating system or its compilers/interpreters.

Your computer can carry out extended precision computations if you simply teach it to. Consider the problem of computing powers of two up to N, where N is very large. A simple program comes to mind:

```
5 INPUT N
10 PRINT 2↑N
20 GO TO 5
30 END
```

Most machines might allow you to go as high as 2^{26} which is 67,108,864. For 2^{27} which is 134,217,720 you would probably have to settle for 1.342177E+8. From there on you can forget about accuracy, you'll have to settle for being "in the ballpark."

Think About Doubling

If we *think* about the process of doubling though, we should be able to store the individual digits of succeeding powers of two in an array. It's a simple matter to double the digits in an array *A* and store them in *B*. If the digit is greater than 4, we must also carry a 1 into the next place, since 2 times a number greater than 4 has two digits, the first always a 1.

PROGRAM 2 below will print out powers of two until the "cow's come home,"—actually until the value has more than 500 digits. The *A* array is multiplied digit-for-digit by 2, then the carry *C* is added to produce a digit for the *B* array. Lines 120-160 do the computation. Line 140 computes the product and adds the carry *C*. Line 150 sets the carry for the next digit. If a power of two has *D* digits, a counter is bumped in line 125 to keep track of how many digits are in the final answer. Leading zeros are suppressed by making the contents of an array negative until something is stuffed into it—lines 165 and 167.

PROGRAM #2

```
40 DIM A(500),B(500)
50 A(1)=1
60 A(2)=-1
100 L=3
105 M=1
110 C=D=0
120 FOR K=1 TO L
125 D=D+1
130 IF A(K)<0 THEN 165
140 B(K)=MOD(A(K)*2,10)+C
150 IF A(K)>4 THEN C=1 ELSE C=0
160 NEXT K
165 IF C=0 THEN B(K+1)=-1 ELSE B(K)=1
167 IF C=1 THEN B(K+1)=-1
200 PRINT '2 **';M;' IS'
210 FOR G=D+1 TO 1 STEP -1
220 A(G)=B(G)
240 NEXT G
250 FOR F=D TO 1 STEP -1
260 IF B(F)<0 THEN 275
270 IF B(F)=0 THEN PRINT ' ';B(F);
    ELSE PRINT B(F);
275 NEXT F
280 PRINT
300 L=D+1
305 M=M+1
307 IF M>250 THEN STOP
310 GO TO 110
400 END
```

Some implementations of BASIC print zeros without a leading plus sign—the spaces between numbers in the printout are actually supressed plus signs. Zero doesn't have a sign. To prevent running significant zeros into the other digits line 270 puts a space in front of embedded zeros.

Now for Division

So much for multiplication! Let's look at a division. I got into this problem while teaching about rational numbers. Fractions either terminate or repeat when expressed as decimals. That's right, every rational number is either a terminating decimal or it repeats at some point. To demonstrate we must be able to express these fractions with more precision than normal computer accuracy permits. The solution is to teach the computer to do long division. Forget about successive subtraction, it's the hard way to divide.

You remember long division with decimal points, and trial divisors and all that. The program below will compute any fraction's decimal equivalent until it terminates or repeats—it'll even say how many places it took to repeat. It sometimes requires patience for the repetition to begin. Did you know that 1/4097 repeats only after 4096 places. The program is simplified by the fact we only compute denominators under 1, hence the presence of 10s in lines 160 and 180. A simple modification would allow all possible numerators with the chosen denominator—although mathematicians amongst the readers will recognize that the number of places in the decimal expansion will remain the same. The program can really be reduced to lines 160 and 180 where the divisions are done. The rest of the logic is for orderly printing and place counting. If you don't have a MOD function—R=MOD(N,D) places the remainder from N/D into R—use R=N-INT(N/D)*D.

PROGRAM #3

```
100 X=1
110 DIM A(100),N(100)
120 PRINT
130 'INPUT NUMERATOR AND DENOMINATOR';
140 INPUT N(X),D
```

Stephen Rogowski, 6 Edward St., Cohoes, NY 12047

```
150 PRINT N(X);'/'D;' = .';
160 A(X)=INT((10*N(X))/D)
170 PRINT A(X);
180 N(X+1)=MOD(10*N(X),D)
190 IF N(X+1)=0 THEN PRINT ELSE 230
200 PRINT '(TERMINATES AFTER'X;
210 IF X=1 THEN ' DIGIT.)' ELSE ' DIGITS.)'
220 GO TO 100
230 X=X+1
240 FOR  Y=1 TO X-1
250 IF N(X)=N(Y) THEN 260 ELSE 300
260 PRINT
270 PRINT '(REPEATS AFTER';X-1;
280 IF X-1=1 THEN ' DIGIT.)' ELSE ' DIGITS.)'
290 GO TO 100
300 NEXT Y
310 GO TO 160
320 END
```

Close scrutiny of the results produces some very interesting properties. The list below was produced by a slightly modified (and simplified) version of PROGRAM 3. It is listed below in PROGRAM 4. Notice that there are 16 separate fractional equivalents for fractions with denominators of 17. The expansion has 16 places. Each expansion is a cyclic permutation of the first, i.e. the order of the digits never changes they just start at a different but predictable point. Since 05 is the smallest possible starting point, it must be assigned to 1/17. The next smallest sequence begins with 11, so it must be assigned to 2/17, and so on.

PROGRAM #4

```
100 PRINT 'INPUT DENOMINATOR'
110 INPUT N
120 FOR T=1 TO N-1
130 LET X=0
140 PRINT
150 LET A=T
160 PRINT A;'/';N;' = .';
170 IF (A*10)/N<1 THEN 240
180 LET A=A*10
190 PRINT INT(A/N);
200 LET X=X+1
210 IF X>=N-1 THEN 290
220 LET A=A-INT(A/N)*N
230 GO TO 170
240 LET A=A*10
250 PRINT ' 0';
260 LET X=X+1
270 IF X>=N-1 THEN 290
280 GO TO 170
290 NEXT T
300 END
```

Some Observations

Even more interesting is a relationship within each expansion. Split each 16 place expansion in half. Consider the first eight digits in order against the last eight. Notice the first half is the nines complement of the second half—i.e. if you add the first eight digits to the last eight, you'll get all nines. This always happens when the denominator is a prime *N* and the decimal period is *N-1*. Using 1/4097, the first 2048 place produces 2048 nines when added to the second 2048 places. Even when prime denominator expansions contain fewer than *N-1* digits, if the period is even, both the cyclic permutations—several in these cases—and the nines complement properties will appear.

```
 1/17  = .0588235294117647
 2/17  = .1176470588235294
 3/17  = .1764705882352941
 4/17  = .2352941176470588
 5/17  = .2941176470588235
 6/17  = .3529411764705882
 7/17  = .4117647058823529
 8/17  = .4705882352941176
 9/17  = .5294117647058823
10/17  = .5882352941176470
11/17  = .6470588235294117
12/17  = .7058823529411764
13/17  = .7647058823529411
14/17  = .8235294117647058
15/17  = .8823529411764705
16/17  = .9411764705882352
```

No reason to miss the advantage of extended precision because the manufacturer didn't design it in. Program it in! ☐