Appendix L. Multiplying Numbers with Many Digits

Most calculators display ten digits or less. While this is more than enough for most calculations, this limitation prevents you, for example, from finding directly all the digits in the product of two six digit numbers. If you multiply 285694 * 786603 your calculator will display something like 2.247277575E11, which is equivalent to 2.247277575*10^{11} or 224727757500, whereas the answer with all of its digits expressed is 224727757482.

More advanced calculators and computers have programs that will display factors and products containing many more digits. In fact you can use them to obtain the product of hundred-digit factors.

What these programs do is carry out algorithms similar to ours for multiplication but with blocks of digits instead of individual digits. To show what I mean, compare the following multiplications:

First the product of 78 and 65 with individual digits:

\[
\begin{array}{cccc}
7 & 8 & \times & 6 & 5 \\
\times & 6 & 5 & \downarrow & \downarrow & \downarrow \\
3 & 5 & 4 & 0 & 7 & 9 & 0 & 7 & 9 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
7 & 4 & 4 & 2 & 4 & 8 \\
5 & 0 & 4 & 6 & 8 & 5 & 0 & 7 & 0 \\
4 & 9 & 1 & 7 & 0 & 4 & 1 & 0 & 7 & 0 & 5 & 0 & 7 & 0 \\
\end{array}
\]

What I have done here is break this process down into its simple steps. The first step records the products of the digits in their correct locations but without carrying. The second step shows the result after carrying (the carries in red) and with the columns added. The last two steps show the result of carrying in the product until the final answer is achieved.
Now I mirror that processing but with pairs of digits in each location. Here we want the product of 6728 and 5692:

\[
\begin{array}{cc}
67 & 28 \\
* & 56  \\
\hline
61 & 25 \\
64 & 76 \\
\hline
37 & 15 \\
52 & 68 \\
\hline
\end{array}
\]

In this first step the four products are formed (by calculator). For example, 92*28 = 2576. And the products are shifted just as in single digit multiplication.

\[
\begin{array}{cc}
67 & 28 \\
* & 56  \\
\hline
61 & 25 \\
64 & 76 \\
\hline
37 & 15 \\
52 & 68 \\
\hline
37 & 128 & 157 & 76 \\
\hline
\end{array}
\]

Here we have carried the 61 and 25 in the first partial product and the 37 and 15 in the second.

\[
\begin{array}{cc}
67 & 28 \\
* & 56  \\
\hline
61 & 89 & 76 \\
37 & 67 & 68 \\
\hline
37 & 128 & 157 & 76 \\
\hline
\end{array}
\]

Here the resulting pairs of digits have been aligned and added. And finally the ones in 128 and 157 are carried in the product:

\[
\begin{array}{cc}
67 & 28 \\
* & 56  \\
\hline
61 & 89 & 76 \\
37 & 67 & 68 \\
\hline
38 & 29 & 57 & 76 \\
\hline
\end{array}
\]

to give us our final product 38,295,776.
Here is a 27-line program to carry out this process for two factors, each with many digits. It does so by storing numbers four digits at a time in Matrix A, which must be established separately to contain three rows and at least M+N-1 columns, with M and N representing the number of groups in the two factors. The steps are numbered for reference in the discussion that follows:

```
PROGRAM:BIGMULT
1  "M"?→M
2  "N"?→N
3  M+N-1→T
4  For 1→I To M
5    "X"?→Mat A[1,I]
6    Next
7  For 1→I To N
8    "Y"?→Mat A[2,I]
9    Next
10 For 1→I To T
11    0→Mat A[3,I]
12    Next
13 For 1→J To M
14    For 1→K To N
15      J+K-1→I
17      Next
18    Next
19  For T→I To 2 Step -1
20    Intg(Mat A[3,I]×10^-4)→C
23    Next
24  For 1→I To T
25    Mat A[3,I],
26    Next
27  Stop
```

1 On the TI-84 calculator the product can have up to 200 digits.
To show how this program works I will consider how the product of two large numbers would be processed. If, for example, we wanted to find the product of the 17-digit number, 35448796577934447, and the 15-digit number, 890038376249965, we would first break the factors down into 4-digit groups, working right to left:

35448796577934447 becomes 3 5448 7965 7793 4447

and

890038376249965 becomes 890 0383 7624 9965.

You now count the number of groups in each factor. The first factor has five groups and the second four. These are the values of M and N requested when you run the program. Once you have entered those two values, you are asked to enter the blocks in each number, left to right. Here is the way the steps would appear when you have correctly entered these factors:

M? 5
N? 4
X? 3
X? 5448
X? 7965
X? 7793
X? 4447
Y? 890
Y? 383
Y? 7624
Y? 9965

If you do that correctly the program should begin to display the product in 4-digit blocks: 3155, 789, 3462, 4009, 1078, 9414, 5604, 4355. (Press **EXE** after you record each four digit block.) Being careful to fill out any block with less than four digits (the 789 becomes 0789) you have the product reported four digits at a time:
I will describe in only general terms how this program works.

You must separately have established Matrix A with 3 rows and at least $M+N-1$ columns,\(^2\) the number of columns determined by the possible size of the product. In our example with $M = 5$ and $N = 4$, we need a 3 by 8 matrix, in the program designated **Mat A**. You can think of it as something like this:

```
  ____  ____  ____  ____  ____  ____  ____  ____
  ____  ____  ____  ____  ____  ____  ____  ____
  ____  ____  ____  ____  ____  ____  ____  ____
```

Lines 4–6 enter the first factor groups in the first line and lines 7–9 the second factor groups in the second line:

```
  3  5448  7965  7793  4447  ____  ____  ____
  890  383  7624  9965  ____  ____  ____  ____
  ____  ____  ____  ____  ____  ____  ____  ____
```

Lines 10–12 enter 0s in the 3rd line in case values reside there from earlier programs.

```
  3  5448  7965  7793  4447  ____  ____  ____
  890  383  7624  9965  ____  ____  ____  ____
  0  0  0  0  0  0  0  0  0
```

Lines 13 to 18 calculate the partial products and add them, recording the sums in row 3. Unlike our paper and pencil algorithm, these partial products and their sums are calculated left to right. This may be done because any necessary carrying will be done in a separate series of steps. Notice that these sums now each have up to nine digits. Here is the way that third row would now look:

```
  ____  ____  ____  ____  ____  ____  ____  ____
```

\(^2\) For how to enter a matrix in a program, see Appendix A.
Lines 19-23 take care of the carrying in order to leave only four digits in each column except the leftmost. Note first that this section of the program works from right to left, the \texttt{1} in the instruction \texttt{For(I,T,1,\texttt{-1})} indicating that the increment is \texttt{-1} rather than \texttt{+1}.

You can see how this is done by considering how that rightmost entry, 44314355 is processed. First, in line 20 it is divided by $10^4$. This gives 4431.4355 and \texttt{C} is the integer value of this number, 4431. In line 21 that number is multiplied by $10^4$ to produce 44310000 and that number is subtracted from the original 44314355 leaving the desired 4-digit number, 4355. In line 22, \texttt{C} (still 4431) is added to the next value to the left to change it from 111561173 to 111565604.

Lines 24 to 26 display the final answer in groups of four digits.

**Checking Your Program**

It is always good procedure to check to see if your programs produce correct results. You could, of course, check the program offered here by using the same numbers that served as an example. But these numbers might have been wrong.

An example like $1,000,000,000 \times 100,000,000$ is not especially useful because, even if the program did give the correct answer, it would not have involved any carrying and thus would not test parts of the program. Instead, I suggest an example like: 999,999,999,999 \times 999,999,999,999. This will test (quite severely) those carrying properties, and we can still work out the product for comparison.

$$999,999,999,999 \times 999,999,999,999$$

is equivalent to:
(1,000,00000,0000 - 1)^2

Squaring, we obtain:

1,000,000,000,000^2 - 2*1,000,000,000,000 + 1

which is equivalent to:

\[
\frac{1,000,000,000,000,000,000,000,000,000,000,001}{2,000,000,000,000,000,000}
\]

with the result:

999,999,999,999,998,000,000,000,001

This is a nice result to check against your program. Your program should print out:

99999999
9998
0
0
1