BASIC ARITHMETIC ON THE MICRO*

Peng Tsu Ann
National University of Singapore

We start off by looking at the way numbers are represented by 0 and 1 in the binary system. We are concerned here only with the number of binary digits required to represent a positive integer. In the decimal system we use the digits 0, 1, 2, ..., 9 to make up a positive integer of any magnitude. In the binary system we use only the digits 0, 1. The number of digits needed in the binary representation of an integer is greater than that in the decimal system. The question is how much greater. We know that the numbers 0 and 1 can be represented using one digit, i.e.

\[
\begin{align*}
  &0 \\
  &1
\end{align*}
\]

The numbers 0, 1, 2, 3 can be represented using two digits, i.e.

\[
\begin{align*}
  00 &\quad 01 \\
  0 &\quad 1
\end{align*}
\]

The numbers 0, 1, 2, ..., 7 can be represented by using three digits, i.e.

\[
\begin{align*}
  000 &\quad 001 &\quad 010 &\quad 011 &\quad 100 &\quad 101 &\quad 110 &\quad 111 \\
  0 &\quad 1 &\quad 2 &\quad 3 &\quad 4 &\quad 5 &\quad 6 &\quad 7
\end{align*}
\]

The numbers 0, 1, 2, ..., 15 can be represented by using four digits. For example,

\[
\begin{align*}
  1011 &\quad 1111 \\
  11 &\quad 15
\end{align*}
\]

If we want to represent the numbers 0, 1, 2, ..., 9, how many binary digits do we need? This number \(x\) can be obtained from the equation

\[2^x = 10.\]

A simple calculation will give \(x \approx 3.3\). So we need a little more than 3 digits to represent the 10 numbers 0, 1, 2, ..., 9.

Now let us go over to the computer. Most microcomputers use 2 bytes (1 byte = 8 bits or 8 binary digits) to represent integers. The number of integers that can be represented is therefore \(2^{16} = 65536\). One half of these numbers are negative and the other half non-negative. This means the range of integers that can be used in integer calculations is restricted to \(-32768\) to \(32767\).

*Text of the Presidential Address delivered to the Singapore Mathematical Society on 25 March 1983.
In real number computations a typical microcomputer (not the Apple) uses 4 bytes (= 32 digits) to represent a number, but not all the bits are used to represent the "significant" part of the number. One byte is used to represent the exponent e.g. a real number (single-precision) of the form

\[ 2.34567 \times 10^4 \]

will appear as

\[ 2.34567E + 4 \text{ or } 0.234567E + 5 \]

in most single-precision representation. How many digits can we get?

Answer: approximately \( \frac{24}{3.3} \approx 7 \). But usually only 6 digits are displayed.

Thus we see that using integers only a number must not exceed 32767 and using real numbers it must not exceed 999999 (actually you can exceed this but to be on the safe (accurate) side, use this as your limit.) What do we do if we want to multiply two six-digit numbers (such as 878432 and 610472) and to get the exact answer? If you have an Apple or some other inexpensive microcomputer, there isn't much you can do if you leave everything to the machine (i.e. if you want just to type in the numbers and leave the machine to work it out). If you have a more powerful microcomputer you will get double-precision arithmetic which uses 8 bytes and gives you 16-digit accuracy. Why 16 digits? Answer: \( 7 + \frac{32}{3.3} \approx 17 \).

(If you want a better explanation consult people in the Computer Science Department or the Electrical Engineering Department. I am out of my depth here.) But then what do you do when you want the exact answer when two 10-digit integers are multiplied? Here is where we actually begin our talk.

We shall begin with addition. Since even with the double precision capacity the microcomputer cannot handle integers with more than 16 digits, how do we enter integers which have 20 digits? The short answer is "Use strings". Here I must assume that the word "string" is meaningful to you.

We shall now try to explain the operations of addition, subtraction, multiplication and division.

Addition

To add two integers \( A \) and \( B \) (say each of 20 digits) we write them as

\[
\begin{align*}
A(20)A(19) & \ldots \ldots \ldots A(3)A(2)A(1), \\
B(20)B(19) & \ldots \ldots \ldots B(3)B(2)B(1),
\end{align*}
\]

where \( A(i) \) is the \( i \)th digit of \( A \) counting from the right, etc. The sum, \( C \), has 20 or 21 digits and we write it as

\[
\begin{align*}
C(21)C(20)C(19) & \ldots \ldots \ldots C(3)C(2)C(1).
\end{align*}
\]
To get $C(i)$ for $i = 1, \ldots, 20$ we proceed as we do by hand:

If $A(i) + B(i) < 10$, then put $C(i) = A(i) + B(i)$;
if $A(i) + B(i) \geq 10$, then put $C(i) = A(i) + B(i) - 10$
and replace $A(i + 1)$ by $A(i + 1) + 1$.

Put $C(21) = A(21)$.

In BASIC this can be written as

```
10 FOR I = 1 TO 20
20 C(I) = A(I) + B(I)
30 IF C(I) > 9 THEN C(I) = C(I) - 10: A(I+1) = A(I+1) + 1
40 NEXT I
```

We use the same notation as for addition. We assume that $A \geq B$. To get $C(i)$ for $i = 1, \ldots, 19$ we also proceed as we do by hand:

If $A(i) \geq B(i)$, then put $C(i) = A(i) - B(i)$;
if $A(i) < B(i)$, then put $C(i) = 10 + A(i) - B(i)$
and replace $A(i + 1)$ by $A(i + 1) - 1$.

Put $C(20) = A(20) - B(20)$.

Multiplication

We illustrate the method by an example. Suppose that we want to find the product of 234 and 56. Let us first perform the multiplication by hand:

```
234
x 56
1404
1170
13104
```

To be able to do this we need to know our 9 x 9 table. Suppose that we know our 9 x 99 table we can do the above multiplication as follows:

```
234
x 56
224
168
112
13104
```

The second method seems more complicated than the first and requires more steps, but it is easier and faster to implement on a computer. Let us see how this can be carried out step by step on a computer.
Let \( A = A(3)A(2)A(1) \) and let \( B = 56 \). The product \( A(3)A(2)A(1) \times B \) has 5 digits; let us call them \( A'(5)A'(4)A'(3)A'(2)A'(1) \).

To get \( A'(1) \):

\[
A(1) \times B = 4 \times 56 = 224 \\
C(1) = \text{INT}(224/10) = 22 \\
A'(1) = A(1) \times B - 10 \times C(1) = 4
\]

The integer \( C(1) \) is to be carried forward.

To get \( A'(2) \):

\[
A(2) \times B = 3 \times 56 = 168 \\
A(2) \times B + C(1) = 168 + 22 = 190 \\
C(2) = \text{INT}(190/10) = 19 \\
A'(2) = 190 - 10 \times C(2) = 0
\]

To get \( A'(3) \):

\[
A(3) \times B = 2 \times 56 = 112 \\
A(3) \times B + C(2) = 112 + 19 = 131 \\
C(3) = \text{INT}(131/10) = 13 \\
A'(3) = 131 - 10 \times C(3) = 1
\]

To get \( A'(4), A'(5) \):

If we put \( A(4) = 0, A(5) = 0 \) and go through the process we get

\[
A'(4) = 3 \\
A'(5) = 1
\]

In fact, there is no need to find \( A'(4), A'(5) \) separately because \( A'(5)A'(4) = C(3) \).

In BASIC we can program the above steps as follows

```
10 FOR I = 1 TO 3
20 A(I) = A(I) * B + C
30 C = INT(A(I)/10)
40 A(I) = A(I) - 10 * C
50 NEXT I
60 A(4) = C.
```

This shows why the second method of multiplying two integers is easier to implement on a computer. But there is one problem. What happens if \( B \) is very large? Answer: use the first method.
Division

As for multiplication we use an example to illustrate the method. What we actually do is long division on a computer. Let us see how we divide 692653 by 345 by hand:

\[
\begin{array}{c|c}
345 & 692653 \\
\hline
2007 & \\
692 & \\
0 & \\
265 & \\
2415 & \\
238 & \\
\end{array}
\]

Let us analyse our steps. Step 1: we take the first 3 digits of 692653 because the divisor has 3 digits. Step 2: we divide 692 by 345 to get the quotient 2 (this may not be easy and involves guesswork). Step 3: we do a multiplication and then a subtraction. Step 4: we go back to Step 1 to repeat the process.

It appears that there is no problem here, but if we look a bit closer we see that what we have actually done was not quite straightforward. In Step 4, we repeated Step 1 to get the quotient. Since 265 < 345, the quotient is 0. But if we had put in "00" instead of "00" in the quotient we would have got a wrong answer. So we need to be careful here. To carry out the above division on a computer we have to tell the computer exactly what to do and not just "you know what I mean" or "it is clear".

There is a more straightforward way of doing division. The method can be easily programmed using BASIC. The programming is left to the reader. We use the same example.

\[
\begin{array}{c|c}
345 & 692653 \\
\hline
002007 & \\
692 & \\
0 & \\
265 & \\
0 & \\
238 & \\
\end{array}
\]

The method is clear. More computations are needed and so in some cases it is slower than the first method.
Using the built-in double precision arithmetic capacity of some micros we can
divide a 16-digit integer by a 15 digit integer rather quickly. With a bit of pro-
gramming and care we can divide an integer of any number of digits by an integer
of at most 15 digits. What if the divisor has more than 15 digits? The principle is
the same but the implementation requires a lot more work. In fact, it can be a
challenge to produce an efficient program for division on a micro. I know because
I have tried.

We shall illustrate the techniques explained in this talk with some sample
programs written in Microsoft BASIC.

EXACT MULTIPLICATION

```
10 REM EXACT MULTIPLICATION
20 PRINT
30 PRINT TAB (30) “EXACT MULTIPLICATION”
40 PRINT : PRINT
50 INPUT “FIRST INTEGER”; A$
60 INPUT “SECOND INTEGER”; B$
70 PRINT
80 DEFINT A-D, I-M
90 DIM A(250), B(250), D(500)
100 LA = LEN(A$) : LB = LEN(B$) : L = LA + LB
110 FOR J = 1 TO LA
120 A(J) = VAL(MID$(A$LA-J+1,1))
130 NEXT J
140 FOR J = 1 TO LB
150 B(J) = VAL(MID$(B$LB-J+1,1))
160 NEXT J
170 FOR I = 1 TO LB
180 FOR J = 1 TO LA + 1
190 H = J + I - 1
200 D(H) = D(H) + A(J) * B(I) + C
210 C = INT(D(H)/10)
220 D(H) = D(H) - 10 * C
230 NEXT J
240 NEXT I
250 IF D(L) = 0 THEN L = L - 1
260 PRINT “THE PRODUCT IS” TAB(17);
270 FOR K = L TO 1 STEP -1
280 PRINT MID$(STR$(D(K)), 2);
290 NEXT K
300 PRINT : PRINT
310 END
```
10 REM EXACT POWERS
20 PRINT
30 PRINT TAB(30) "EXACT POWERS"
40 PRINT : PRINT
50 DEFINT A-D, I-N, P
60 INPUT "ENTER INTEGER (BASE)", B$
70 INPUT "MAXIMUM EXPONENT"; N
80 PRINT
90 DIM A(500), B(100), D(500)
100 PRINT "EXPONENT POWER OF" B$
110 PRINT
120 PRINT 1 TAB(11) B$
130 LB = LEN(B$): LA = LB: L = LA + LB
140 FOR J = 1 TO LB
150 B(J) = VAL(MID$(B$, LB - J + 1, 1))
160 A(J) = B(J)
170 NEXT J
180 FOR M = 2 TO N
190 FOR I = 1 TO LB
200 FOR J = 1 TO LA + 1
210 H = I + J - 1
220 D(H) = D(H) + A(J) * B(I) + C
230 C = INT(D(H)/10)
240 D(H) = D(H) - 10 * C
250 NEXT J
260 NEXT I
270 NEXT M
280 FOR K = 1 TO L
290 A(K) = D(K)
300 D(K) = 0
310 NEXT K
320 PRINT TAB(11); M
330 FOR K = L TO 1 STEP -1
340 PRINT MID$(STR$(A(K)), 2)
350 NEXT K
360 LA = L: L = LA + LB
370 PRINT
380 NEXT M
390 PRINT
400 END
EXACT DIVISION

10 REM DIVISION BY DIVISOR OF NOT MORE THAN 15 DIGITS
20 PRINT
30 DEFDBL A-C, R : DEFINT J, L
40 INPUT "ENTER POSITIVE INTEGER ", A$
50 INPUT "ENTER DIVISOR (NOT MORE THAN 15 DIGITS) ", B$
60 PRINT
70 LB = LEN(B$) : B = VAL (B$)
80 A1$ = LEFT$(A$, 16) : LA1 = LEN(A1$)
90 A = VAL(A1$)
100 C = INT(A/B)
110 R = A-C*B
120 Q1$ = MID$(STR$(C), 2) : LQ1 = LEN(Q1$)
130 J = J + 1
140 IF J = 1 THEN Q$ = Q1$ : GOTO 170
150 Q2$ = STRING$(LA1-LR-LQ1, 48)
160 Q$ = Q$+Q2$+Q1$
170 R$ = MID$(STR$(R), 2)
180 LR = LEN(R$)
190 A$ = R$+MID$(A$, 17)
200 LA = LEN(A$)
210 IF LA < LB OR (LA = LB AND A$ < B$) THEN GOTO 230
220 GOTO 80
230 Q$ = Q$+STRING$(LA-LR, 48)
240 WHILE LEFT$(A$, 1) = "0"
250 A$ = MID$(A$, 2)
260 WEND
270 IF A$ = "" THEN A$ = "0"
280 PRINT "QUOTIENT = "; Q$
290 PRINT 
300 PRINT "REMAINDER = "; A$
310 PRINT
320 END
10 REM THE FIBONACCI SEQUENCE
20 PRINT TAB(30) "FIBONACCI SEQUENCE"
30 DEFINT A-C, I--N
40 DIM A(250), B(250), C(251), D$(251)
50 PRINT: PRINT
60 INPUT "HOW MANY TERMS"; N
70 PRINT
80 PRINT "TERM" TAB(11) "NUMBER"
90 PRINT
100 A$ = "1" : B$ = "1"
110 PRINT TAB(11) A$
120 PRINT TAB(11) B$
130 FOR M = 3 TO N
140 PRINT M TAB(11);:
150 GOSUB 1000
160 A$ = B$ : B$ = C$
170 PRINT C$
180 NEXT M
190 PRINT
200 END
210 REM ADDITION SUBROUTINE
220 LA = LEN(A$) : LB = LEN(B$)
230 FOR J = 1 TO LA
240 A(J) = VAL(MID$(A$, LA- J + 1, 1))
250 NEXT J
260 FOR J = 1 TO LB
270 B(J) = VAL(MID$(B$, LB- J + 1, 1))
280 NEXT
290 IF LA >= LB THEN L = LA
300 ELSE L = LB
310 FOR I = 1 TO L
320 C(I) = A(I) + B(I)
330 IF C(I) > 9 THEN C(I) = C(I) - 10: A(I+1) = A(I+1) + 1
340 NEXT I
350 IF A(L+1) = 1 THEN C(L+1) = 1: L = L + 1
360 FOR K = 1 TO L
370 D$(K) = MID$(STR$(C(K)), 2) + D$(K-1)
380 D$(K - 1) = ""
390 NEXT K
400 C$ = D$(L)
410 FOR K = 1 TO L
420 A(K) = O: B(K) = O: C(K) = 0
430 NEXT K
440 RETURN
BINOMIAL COEFFICIENTS

10 REM BINOMIAL COEFFICIENTS
20 PRINT
30 PRINT "BINOMIAL COEFFICIENTS C(N, K)"
40 PRINT : PRINT
50 DEFINT I-N, X-Z
60 DEFFDBL A-C, E-G, R
70 DIM X(251), Z$(251)
80 INPUT "POSITIVE INTEGER N"; N
90 INPUT "POSITIVE INTEGER K"; K
100 PRINT
110 IF N < 0 OR K < 0 THEN PRINT "ENTER POSITIVE INTEGERS" : PRINT: GOTO 80
120 IF N < K THEN PRINT "ENTER N, K WITH N GREATER OR EQUAL TO K" : PRINT : GOTO 80
130 E$ = "1"
140 PRINT "C("MID$(STR$(N), 2)"" , ""0"" ); E$ = 1"
150 FORM= 1 TO K
160 IF LEN(E$) < 15 THEN E = VAL(E$) : GOTO 200
170 X$ = E$: Y = N - M + 1 : GOSUB 3000
180 A$ = Z$: B = M : GOSUB 4000
190 E$ = Q$: GOTO 220
200 F = N - M + 1 : G = M : E = E * F/G
210 E$ = MID$(STR$(E), 2)
220 IF M > 9 THEN GOTO 250
230 PRINT "C("MID$(STR$(N), 2)" ; MID$(STR$(M), 2)" E$ = " E$"
240 GOTO 260
250 PRINT "C("MID$(STR$(N), 2)" ; MID$(STR$(M), 2)" E$ = " E$"
260 NEXT M
270 PRINT
280 END

3000 REM MULTIPLICATION SUBROUTINE
3010 LX = LEN (X$)
3020 FOR I = 1 TO LX
3030 X(I) = VAL(MID$(X$, LX - 1 + 1, 1))
3040 NEXT I
3050 Z = 0
3060 FOR J = 1 TO LX
3070 X(J) = X(J) * Y + Z
3080 Z = INT(X(J)/10)
3090 X(J) = X(J) - 10 * Z
3100 NEXT J
3110 X(LX + 1) = Z
3120 IF Z <> 0 THEN LX = LX + 1
3130 FOR J = 1 TO LX
**BINOMIAL COEFFICIENTS (CONTINUED)**

3140 \[ Z(J) = \text{MID}(\text{STR}(X(J)), 2) + Z(J - 1) \]
3150 \[ Z(J - 1) = "" \]
3160 NEXT J
3170 Z$ = Z$(LX)
3180 RETURN

**REM DIVISION SUBROUTINE**

4010 J = 0
4020 A1$ = LEFT$(A$, 16) : LA1 = LEN(A1$)
4030 A = VAL(A1$)
4040 C = INT(A/B)
4050 R = A - C \times B
4060 Q1$ = MID$(\text{STR}(C), 2) : LQ1 = \text{LEN}(Q1$)
4070 J = J + 1
4080 IF J = 1 THEN Q$ = Q1$ : GOTO 4110
4090 Q2$ = STRING$(LA1-LR-LQ1, 48)
4100 Q$ = Q$ + Q2$ + Q1$
4110 R$ = MID$(\text{STR}(R), 2)$
4120 LR = LEN(R$)
4130 A$ = R$ + MID$(A$, 17)
4140 LA = LEN(A$) : B$ = MID$(\text{STR}(B), 2) : LB = \text{LEN}(B$)
4150 IF LA < LB OR (LA = LB AND A$ < B$) THEN GOTO 4170
4160 GOTO 4020
4170 Q$ = Q$ + STRING$(LA-LR, 48)
4180 RETURN

The above programs should run without change on any microcomputer which supports Microsoft BASIC (also known as MBASIC or BASIC-80). On machines that run their own versions of Microsoft BASIC some modifications may be necessary (e.g. in the use of the string function MID$). The dimension statements in the programs can be changed to save memory or to increase accuracy. All the programs can be compiled to increase the speed of computation. For the mathematically inclined it is fun to experiment with large numbers on a microcomputer; it is often more satisfying than playing games and certainly more interesting than doing accounts payable.