Table 1, p. 15–16, *American Practical Navigator* (originally by N. BOWDITCH), U. S. Hydrographic Office, no. 9, revised edition of 1938, is for Radio Bearing Conversion. For difference of longitude = 1°(0°.5)16°.5 and middle latitude 4°(1°)60° the table gives the correction to be applied to radio bearing to convert to Mercator bearing. This table is to be replaced by the much more extensive new table.

With the new table

(1) Great circle directions can easily be converted to rhumb line directions for plotting radio bearing on a Mercator chart. Since radio waves travel along great circles, such corrections are necessary.

(2) Rhumb lines may be converted to great circle directions. Usually tangents or chords of the great circle, for about 5° of longitude, are sailed. The Table will be of value in planning courses, and will quickly show when the difference between rhumb line courses and great circle courses is significant.

If values are desired within the 5° intervals of the table at higher latitudes they may readily be found by double interpolation. The table may be of service in both marine and air navigation.

FRANCES W. WRIGHT

Harvard College Observatory

AUTOMATIC COMPUTING MACHINERY

Edited by the Staff of the Machine Development Laboratory of the National Bureau of Standards. Correspondence regarding the Section should be directed to Dr. E. W. CANNON, 418 South Building, National Bureau of Standards, Washington 25, D. C.

TECHNICAL DEVELOPMENTS

The leading article of this issue of *MTAC*, "A Bell Telephone Laboratories' Computing Machine—I," by Dr. FRANZ L. ALT, is our current contribution under this heading.

DISCUSSIONS

Decimal Point Location in Computing Machines

General. For various reasons, the majority of large scale automatic-sequence digital computers in existence or in the design stage use fixed decimal (or binary) point numbers. In all the machines of which the writer has knowledge, the decimal (or binary) point is located at the extreme left so as to make all numbers fall in the range -1 to $+1$. It is the writer's opinion that this choice is not the best, and that location of the decimal (or binary) point several digits away from the extreme left is better in almost every respect. Some of the considerations leading to this opinion are presented in this paper. For the sake of simplicity and definiteness, unless otherwise stated, the discussion refers to a machine in which data are stored in some form of memory which restricts numbers to a specified fixed number of decimal digits. This machine is assumed capable of automatically following a prescribed sequence of instructions which include addition, subtraction, multiplication, division, and transfer operations. The results of the operations are rounded off on the right to the same number of digits as the original data, after positioning the digits in accordance with a single fixed decimal point location. It is further assumed that the machine has no provision for handling excess digits on the left (exceeding capacity of the machine) created by any operation, so that such an occurrence implies an error in programming.

Possibility of Exceeding Capacity. One of the chief advantages claimed for the extreme-left-hand position of the decimal point is that the machine capacity as regards size of individual numbers cannot be exceeded in multiplication, since a product of two numbers, each less than unity, must also be less than unity. As this is not true for the other operations, such as addition or division, this feature would not appear to have much value, since magnitudes of numbers at each step have to be studied by the programmer in any case to avoid exceeding capacity in other operations. Even in multiplication, magnitudes must be considered to correct for the opposite effect caused by this choice of decimal position (the continual decrease in magnitudes of numbers as a result of successive multiplications) by judicious shifting (or division by powers of 10) at suitable stages in the computation.

Although any choice of the decimal point further to the right creates the possibility of obtaining products larger than either factor, thus exceeding limits if the factors are sufficiently large, this is compensated somewhat by the possibility of arranging values (for example by keeping them approximately equal to unity) so that any number of successive multiplications may be programmed without either exceeding limits or losing many significant digits. Furthermore, in division, use of wholly fractional numbers requires that the numerator be smaller than the denominator to keep the quotient within bounds, while much broader limits are permissible if one or more digits are available to the left of the decimal point. The extra magnitude of the checking in division necessitated by choice of the extreme-left decimal point would appear to offset any possible gain in checking of multiplications.

Inherent Accuracy. Another important advantage claimed for use of the extreme-left decimal point location is that this permits maintenance of the greatest number of significant figures in each factor in multiplication and hence leads to greatest accuracy. That this contention cannot be wholly true is easily seen from the following 3-digit example, which compares the result obtained when the decimal point is at the extreme left with that obtained when the decimal point is one digit to the right.

$$.123 \times .111 = .014 \qquad \text{accuracy 2 digits}$$
$$1.23 \times 1.11 = 1.37 \qquad \text{accuracy 3 digits.}$$

Let us examine the question of accuracy in a little more detail. We restrict ourselves to consideration of errors due to the limited number of digits (round-off errors) only and, for simplicity, evaluate the maximum such error instead of the most probable value. We further assume that the programmer has accurate cognizance of the magnitudes of all numbers and can shift them to the most advantageous digital position, subject to the number limitations of the machine, but that no auxiliary operations such as division or multiplication by factors other than powers of 10 are to be employed, so that there is no change in the digits themselves. If numbers have n total digits and the decimal point is m digits from the left, the maximum round-off error in a number of magnitude N is $10^m K/N$ of the number, where

$$K = \tfrac{1}{2} \times 10^{-n}$$

is the error for maximum N. If two numbers M and N are multiplied together, their product will be $P = MN$. The percentage error in P is the

sum of the percentage errors in M and N plus additional round-off error in limiting P to the specified number of digits, making the maximum error in P equal to

$$E = 10^m(1/M + 1/N + 1/P)K,$$

where all values are assumed positive.

For a machine with decimal point at the left, $m = 0$ and M, N, and P must not exceed unity. Minimum E occurs for numbers close to this limit and has the value

$$E \text{ (minimum for } m = 0) = 3K.$$

Maximum E occurs when M and N (and hence P) are smallest. Since, under our assumptions, we may always shift numbers so that the first digit is not a zero, we can insure that M and N are at least .1. For this limit, we have $P = .01$ so that

$$E \text{ (maximum for } m = 0) = (10 + 10 + 100)K = 120K.$$

For a machine with decimal point one digit to the right, $m = 1$, and M, N, and P must not exceed 10. Minimum E occurs for $M = N = \sqrt{10}$ and has the value

$$E \text{ (minimum for } m = 1) = 10(1/\sqrt{10} + 1/\sqrt{10} + 1/10)K = 7.3K.$$

Maximum E occurs when M and N (and hence P) are smallest. Whenever possible, we shift M and N so that the first digit in each is not a zero. When this causes P to exceed 10, we shift the number (say M) with smaller initial digits to the extreme left and allow a single zero digit at the left of the other. With this arrangement, which is always possible, the largest E occurs for $M = \sqrt{10}$, $N = 1/\sqrt{10}$, $P = 1$ and is

$$E \text{ (maximum for } m = 1) = 10(1/\sqrt{10} + \sqrt{10} + 1)K = 45K.$$

We thus see that the maximum error is actually less in this case by a factor of almost 3 as compared to the case where the decimal point is at the extreme left, although the minimum error is not as low. Even for $m = 2$, we get

$$E \text{ (maximum for } m = 2) = 120K,$$

corresponding to M close to 10 and $N = 1$, so that the extreme error, with the decimal point two digits to the right, is no greater than with decimal point at the extreme left.

The above is based on the assumption that the programmer can predict magnitudes very closely and that, in iterated operations, the range of magnitude variation is small. This is generally not the case, and an appreciable spread of values must be allowed for. If we assume uncertainty in each of the factors in a multiplication over a range of 10 to 1, the programmer cannot assure best placement of digits but only that numbers will be at most one digit away from best positioning. Maximum errors will then occur for M and N both one tenth their previous values. For $m = 0$, this corresponds to $M = N = .01$; $P = .0001$, giving

$$E \text{ } (m = 0, \text{ 10 to 1 range}) = (100 + 100 + 10000)K = 10200K.$$

For $m = 1$, the extreme occurs when $M = 1/\sqrt{10}$; $N = 1/10\sqrt{10}$; $P = 0.01$ and is

$$E\ (m = 1,\ 10 \text{ to } 1 \text{ range}) = 10(\sqrt{10} + 10\sqrt{10} + 100)K = 1350K,$$

being almost 10 times as good as the preceding. Similarly, for $m = 2$, $M = 1$; $N = 0.1$; $P = 0.1$ is the extreme case and

$$E\ (m = 2,\ 10 \text{ to } 1 \text{ range}) = 100(1 + 10 + 10) = 2100K.$$

Even for $m = 4$, the extreme is for $M = 10$; $N = 1$, $P = 10$ and has the value

$$E\ (m = 4,\ 10 \text{ to } 1 \text{ range}) = 10000(.1 + 1 + .1) = 12000K,$$

which is not materially greater than for $m = 0$.

Obviously, the greater the range of values in an iterated operation, or the greater the uncertainty in regard to possible magnitudes that may occur, the poorer the choice of decimal point at the extreme left becomes as concerns accuracy, and the greater the number of digits to the left of the decimal point corresponding to minimum extreme error.

Availability of Common Integers and Constants Exceeding Unity. It is certainly highly desirable to have integers such as 1, 2, 3, or 10 and constants such as $\pi = 3.14159$, $e = 2.71828$, $\sqrt{2} = 1.41424$, $\ln 10 = 2.30259$, etc., available for use in programming. With the decimal point at the extreme left, such values cannot be expressed directly but must be divided by a power of 10 or must be given in a different form such as the reciprocal value. This, at best, complicates the programmer's work. As a simple example, consider the usual square root iteration

$$y_{n+1} = (y_n + x/y_n)/2$$

to obtain $y = \sqrt{x}$. The convenient first approximation of $y_0 = 1$ cannot be directly used, nor can the operation be carried out as indicated. It is necessary to change the form to

$$y_{n+1} = \tfrac{1}{2}y_n + \tfrac{1}{2}x/y_n$$

and to take a string of 9's for y_0 to avoid exceeding capacity. This involves one additional initial step (halving of x) and an awkward value for y_0 and still is insufficient if there is any possibility that x itself is a string of 9's, since this would cause y_1 to exceed unity (because of normal round-off's).

Mathematical values do not naturally limit themselves to magnitudes below unity, and such an arbitrary limitation may be expected to lead to various programming difficulties, which may be minimized by allowing some range above unity. The most important value that should be included in the permissible range is unity itself. If, for example, cosine or sine values are required in a computation, it is possible to exceed the capacity of a machine with decimal point at the left if the argument should get too close to zero or $\frac{1}{2}\pi$, requiring either some assurance that this will not occur or sacrifice of a decimal digit for values with the accompanying difficulty of introducing additional factors in the programming. It is certainly far more convenient for the programmer to work with natural values than to keep track of various factors introduced to shift values to lie within range of the machine. Of course, some factors are required with any choice of decimal point loca-

tion, but their number is materially greater for the extreme left location than for locations several digits further to the right. It is only necessary to glance through a mathematical text to verify this statement by observing the relative occurrence of values above, say, 100 as compared to those between 1 and 100.

Problems Involving Extreme Accuracy. A general-purpose computer may be required to solve problems whose solution is desired to greater accuracy than corresponds to the fixed number of digits permitted in a single memory location of the machine. In certain problems, while the solution is not required to excessive accuracy, it is necessary at some intermediate stage to provide much greater precision because of large loss of relative accuracy during the process of computation. Before discussing this matter in regard to decimal machines, it may be noted that, in addition to difficulties similar to those encountered in decimal machines, a binary machine used for problems of extreme accuracy involves questions of precision of conversion between the normally used decimal system and the binary system used by the machine. Since, in general, only integral values convert exactly between these two number systems, a binary machine with binary point at the extreme left requires the programmer to insert powers of 2 as factors as well as powers of 10 in order to avoid conversion errors. For example, 10 would be inserted as $\frac{5}{8}$ with the factor 16 to be remembered by the programmer. While such binary factors can probably be handled expeditiously by high-grade mathematicians, they are certainly more likely to cause errors and require special training for programmers of lower skill.

In order to simplify handling of high-precision problems, we will assume that our machine has available, when called for, a special division process which, in addition to the normal quotient, stores the remainder (shifted to the left up to the decimal point) into a specified memory location. Such a facility as well as a similar one for multiplication will be available in the EDVAC now under construction by the University of Pennsylvania. For a machine of 3 decimal digits with decimal point at the left, this special division will work as follows:

$$.111/.123 = .902 + .000054/.123.$$

The machine will store .902 in one location and .054 in another. For a similar machine with one digit to the left of the decimal point, this operation will be:

$$1.11/1.23 = 0.90 + .0030/1.23,$$

and the machine will store 0.90 in one location and 0.30 in another. In all cases, the second number will always be less than the divisor.

With the aid of this special operation and a corresponding multiplication, and more than one memory location for the results of each operation, almost any degree of accuracy can be achieved. It is to be noted, however, that each value is expressed as a sum of component terms, with suitable factors, each stored as a separate number. Successive operations are, therefore, operations on polynomials and result in similar polynomials. At each stage, care must be taken not to exceed capacity in any term of the polynomial and to "carry over" to the next term any indicated excess. With the decimal point at the extreme left, this is a difficult chore and generally requires

extensive modification of procedure with addition of quite a few comparison instructions (branch orders) to check for and correct this condition without momentarily exceeding capacity at some point. With one or more digits to the left of the decimal point, this problem becomes much simpler to take care of since, as will be noted in later numerical examples, all terms other than the first are normally less than unity. It is thus possible to perform the next operation without considering the exceeding of capacity as a result of it, and then to add the integral portion of each term to the preceding (after suitable shift to the right) as a "carry." In fact, this "carry" process may frequently be postponed until after a number of operations are carried out. For example, the evaluation of e to any number of digits can be completely carried out on a machine of the type we are discussing, with one digit to the left of the decimal point, with only a single "carry" routine in the entire procedure.

It may be worth while to give a simpler numerical example here. Suppose we had a 3-digit machine and wished to evaluate

$$101/301 + 201/401$$

to 7 digits. Consider first the decimal point at the extreme left. Using special division, we obtain

$$\begin{aligned} .101/.301 &= .335 \text{ quotient} + .165 \times 10^{-3} \text{ remainder} \\ .165/.301 &= .548 \text{ quotient} + .052 \times 10^{-3} \text{ remainder} \\ .052/.301 &= .173 \text{ normal quotient} \end{aligned}$$

so that we store 101/301 in 3 memory locations as

$$.335 + .548 \times 10^{-3} + .173 \times 10^{-6}.$$

Similarly, we get

$$201/401 = .501 + .246 \times 10^{-3} + .883 \times 10^{-6}.$$

We cannot now add terms directly since their sums may exceed unity. It is thus necessary to examine values before summing them. One way of doing this is to compare the complement of .173, which is .827, with .883. Since it is smaller than the latter, we subtract it from the latter obtaining .056 and carry 1 to the next term. Comparing the complement of .548, which is .452, with .246, we find it larger, indicating no carry. We thus add the original terms, obtaining .794, to which we must add .001 carry from the previous addition, requiring another comparison to show that we may directly add to get .795 with no carry. The first terms can, of course, be directly added without any ado since we know their order of magnitude. We thus get

$$101/301 + 201/401 = .836 + .795 \times 10^{-3} + .056 \times 10^{-6}.$$

If additional similar terms were to be added, the work would, of course, increase enormously.

Now let us consider the same problem with the decimal point one digit to the right. As before, we get

$$\begin{aligned} 1.01/3.01 &= 0.33 \text{ quotient} + 1.50 \times 10^{-2} \text{ remainder} \\ 1.50/3.01 &= 0.55 \text{ quotient} + 1.45 \times 10^{-2} \text{ remainder} \\ 1.45/3.01 &= 0.48 \text{ quotient} + 0.52 \times 10^{-2} \text{ remainder} \\ 0.52/3.01 &= 0.17 \text{ normal quotient} \end{aligned}$$

so that we store 101/301 in 4 memory locations as

$$0.33 + 0.55 \times 10^{-2} + 0.48 \times 10^{-4} + 0.17 \times 10^{-6}.$$

Similarly, we get

$$201/401 = 0.50 + 0.12 \times 10^{-2} + 0.46 \times 10^{-4} + 0.88 \times 10^{-6}.$$

Here we may add corresponding terms without worry of exceeding capacity. In fact, since each term is less than unity in value and the machine capacity is 10, we could add 10 terms at a time without fear of exceeding limits. We thus get

$$101/301 + 201/401 = 0.83 + 0.67 \times 10^{-2} + 0.94 \times 10^{-4} + 1.05 \times 10^{-6}.$$

We now take the integral part of each term as a "carry" to the preceding term. If less than 9 terms had been previously added, such carry cannot cause excessive magnitudes and can be very simply carried out in succession. In this case, only one carry occurs and we finally get

$$101/301 + 201/401 = 0.83 + 0.67 \times 10^{-2} + 0.95 \times 10^{-4} + 0.05 \times 10^{-6}.$$

The only disadvantage of the new decimal point location is that it requires more terms for equal accuracy; however, the given example accentuates this difficulty because of the small number of total digits and the particular choice of numerical values. For usual machines of 10 or more decimal digits, the increase in number of terms caused by one digit shift of the decimal point would be negligible, but the saving in carry-over difficulties just as large. Choice of numerical values such as 41/301 + 51/401 would further equalize the number of terms.

SAMUEL LUBKIN

Reeves Instrument Corporation
New York City

BIBLIOGRAPHY, Z–II

1. ARTHUR W. BURKS, HERMAN H. GOLDSTINE & JOHN VON NEUMANN, *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument.* Prepared for the Research and Development Service, Ordnance Department, U. S. Army, June, 1946, i, 53 leaves and table, 21.6 × 27.9 cm. Mimeographed.

This report, consisting of 6 sections and a table giving a proposed instruction code, is the first of two papers dealing with "some aspects of the over-all logical considerations arising in connection with electronic computing machines."

Section 1.0, "Principal Components of the Machine," is a statement of the "main organs" a fully automatic general-purpose computing machine must possess. The authors point out that such a machine must have a memory organ, a control organ, an arithmetic organ, and input and output organs.

The computer must be capable of storing tables of functions, the original data of problems, and the intermediate results of computations; it must also be able to store sequences of instructions for performing specified numerical computations. This storage function is performed by a memory organ. The