THE REVIEW OF SCIENTIFIC INSTRUMENTS VOLUME 21, NUMBER 8 AUGUST, 1950

Arithmetic Operations in a Binary Computer

ROBERT F. SHAW
Eckert-Mauchly Computer Corporation, 3747 Ridge Avenue, Philadelphia, Pennsylvania
(Received February 13, 1950)

The application of binary arithmetic in the computing circuits of a high speed digital computer is discussed in detail. The discussion covers, with numerous examples, the use of complements to represent negative numbers, the corrections necessary in the multiplication process as a result of the use of complements, and additional modifications of the process to simplify mechanization. A special division method well-adapted to automatic computer use is described, and round-off procedures are noted briefly. The article is concluded with a discussion of the storage of negative numbers as absolute values with a sign rather than in complement form.

INTRODUCTION

THE extensive development of large-scale digital computers in the past few years has naturally been accompanied by a corresponding development of the mathematical techniques required for the most efficient use of these tools of research. Probably the outstanding feature of electronic circuits, insofar as their application to digital computers is concerned, is their binary nature. It is possible, by relatively simple means, to use these essentially binary circuits for operation in the decimal system, provided the decimal digits are represented in binary coded form. There is little doubt that all significant future development of general purpose computers, particularly for use in the commercial and industrial field, will be concerned with decimal operation. Nevertheless, it is also true that many of the large-scale computers now under development both in this country and abroad are purely binary, at least in their arithmetic operations. The greater simplicity of this type of device and its slightly more efficient use of memory capacity will probably assure it a position of continuing importance in the field of small special-purpose computers such as those contemplated for industrial control applications.

While the present discussion is concerned chiefly with the arithmetic operations of the Binac, a general purpose binary computer recently completed by the Eckert-Mauchly Computer Corporation, they apply to a large degree to most of the other binary computers now under construction. The mathematical developments presented are not intended to be rigorous, but are intended more to convey as simply and clearly as possible the principles underlying the operation of the Binac's arithmetic circuits. For a more thorough and rigorous analysis of these principles, the reader is referred to the report of Burks, Goldstine, and von Neumann, on the computer being built at the Institute for Advanced Study.[1] The arithmetic operations in this computer are similar to those in the Binac, although the latter is serial in nature, while the Institute's computer is of the parallel type.

[1] Burks, Goldstine, and von Neumann, *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument* (Institute for Advanced Study, Princeton, New Jersey, 1947), second edition, Part 1, Vol. 1.

I. RANGE OF NUMBERS

The arithmetic circuits and memory of a computer are designed to handle data in units, each consisting of a fixed number of binary digits; since these units may represent either numbers or instructions,[2] they are commonly referred to as "words." The choice of word length must be a compromise among a number of factors, such as available memory capacity, operating speed, required accuracy of numerical data, and other considerations. A word length of 30 binary digits, equivalent numerically to slightly less than 10 decimal digits, was chosen for the Binac; other binary computers now in the process of construction use word lengths varying from 16 to 44 binary digits.

Placement of the binary point must next be determined. Multiplication will be simplified if the point is placed to the left of the first significant digit, since it will then be possible to form the product of two factors, each having the maximum of 30 significant digits, without exceeding the capacity of the computer, and without resorting to "floating point" methods which require the keeping of a separate record of the binary point position. Placing of the point in this position also leads to a convenient method of handling negative numbers, as will be seen later. The absolute values of numbers which the computer can represent will lie in the range $0 \leqq x < 1$, and, since 30 binary digits can be stored, the smallest increment which the computer can represent is 2^{-30}.

Both positive and negative numbers can be represented if one additional digit, to the left of the binary point, is used. This digit will be a 0 for positive numbers and a 1 for negative numbers; and the absolute value of the latter will actually be replaced by its complement with respect to 1; for example: $+\frac{3}{16}$ will be written 0.0011, but $-\frac{3}{16}$ will be written 1.1101. (It will be noted that if the sign is treated as if it were simply another digit, negative numbers are represented by

[2] Instructions and numerical quantities are handled interchangeably in both the memory and arithmetic circuits, instructions being modified in the latter circuits by addition or subtraction of constants where necessary. However, an instruction differs from a quantity in that it can be interpreted by the control circuits of the computer and through those circuits can cause predetermined operations to be performed.

TABLE I.

Addends	Sum	Carryover
0 and 0	0	0
0 and 1	1	0
1 and 1	0	1

their complements with respect to 2. This statement will be further amplified in the discussion of addition and subtraction.) The total range of numbers which the computer can represent is then $-1 \leqq x < 1$.

II. ADDITION

It will next be shown that the correct algebraic sum of two quantities can be formed by an adder which will form the correct sum of two positive numbers. The basic rules of binary addition are illustrated in Table I. Since the adder must be capable of forming the correct sum even if two ones from the two addends coincide with a carryover from the preceding digit, Table II, which is developed from the basic principles in Table I, summarizes the results which the adder must produce. An adder satisfying these requirements, then, will form the correct sum of two positive numbers. Since it is not the purpose of this paper to discuss circuit details, it is sufficient to state here that these requirements can be met electronically by relatively simple means.

The significance of the statement, previously made, that negative numbers are represented by their complements with respect to 2, must now be examined. This implies that if $-1 \leqq x < 0$, x is actually written $2+x$, which obviously lies in the range between 1 and 2. Thus, in the example previously given, $-\frac{3}{16}$ (in binary form -0.0011) is written 1.1101. If the digits to the left of the sign position are ignored, any positive number $0 \leqq x < 1$ is indistinguishable from $2+x$; for example, $\frac{3}{16}$ in binary form is 0.0011; $\frac{3}{16}+2$ is 10.0011, which becomes 0.0011 again if the 1 to the left of the sign digit is ignored.

Thus, if an adder is constructed which discards carries beyond the sign digit, it should work equally well for positive or negative numbers. The following examples will illustrate this.

0.0011	$\frac{3}{16}$
0.0111	$\frac{7}{16}$
0.1010	$\frac{5}{8}$
0.0011	$\frac{3}{16}$
1.1001	$-\frac{7}{16}$
1.1100	$-\frac{1}{4}$
1.1101	$-\frac{3}{16}$
1.1001	$-\frac{7}{16}$
(1) 1.0110	$-\frac{5}{8}$

TABLE II.

Addends	Sum	Carryover
0, 0, and 0	0	0
0, 0, and 1	1	0
0, 1, and 1	0	1
1, 1, and 1	1	1

(The carryover, shown in parentheses, is discarded.)

1.1101	$-\frac{3}{16}$
0.0111	$\frac{7}{16}$
(1) 0.0100	$\frac{1}{4}$

(Again, the carryover is discarded.)

To summarize these results, then, it is possible to use a simple binary adder to add positive or negative numbers, provided (a) each quantity is confined to the range $-1 \leqq x < 1$ and a 2 is added to the quantity before it is sent into the machine, and (b) any digits to the left of the 2^0 position, called the sign position, are discarded, whether they result from the original addition of 2 to the number or from an addition in the course of subsequent computation.

III. SUBTRACTION

Since the addition process just described works equally well for positive and negative quantities, subtraction can be accomplished by changing the sign of the subtrahend and adding. Since the sign reversal is accomplished by determining the complement of the quantity with respect to 2, a procedure for complementing is required.

First, consider the result obtained by selecting a quantity, changing all its zeros to ones and its ones to zeros, then adding the result to the original number.

$$\begin{array}{r} 0.1101001 \\ 1.0010110 \\ \hline 1.1111111. \end{array}$$

It is evident that the result will always be a series of ones. For n digits to the right of the binary point, the result will be $2-2^{-n}$. The quantity obtained by interchanging zeros and ones is therefore the complement of the original quantity with respect to $2-2^{-n}$; that is, the new quantity $m' = 2-2^{-n}-m$. Since the result required is $2-m$, it is only necessary to add 2^{-n} to the result obtained by interchanging ones and zeros:

$$m = 0.110100$$
$$-m = 1.001011 + 0.000001 = 1.001100.$$

Complementing is thus accomplished by interchanging ones and zeros in the quantity to be complemented and adding a one to the result in the least significant digit position.

A somewhat different subtraction method is dis-

cussed in the section dealing with storage of absolute values.

IV. MULTIPLICATION

Multiplication in the binary system is simplified by the fact that any multiplier digit can be only a one or a zero. This fact results in a process easy to mechanize, since additions and shifts alternate, whereas in the decimal and other number systems, the number of additions between shifts varies in accordance with the multiplier digits. (Of course, the case of no addition can be treated as equivalent to the addition of zero.) Multiplication of two positive binary quantities, then, consists of examining the multiplier digits in succession, starting with the least significant, adding the multiplicand into the partial product if the corresponding multiplier digit is a one, but adding zero if the multiplier digit is zero. After every such addition, the partial product is shifted one place to the right. The following example illustrates the procedure:

0.1101	x	$\frac{13}{16}$
0.1011	y	$\frac{11}{16}$
0.1101	$x\cdot 1$	
0.1101	$x\cdot 1$	
0.0000	$x\cdot 0$	
0.1101	$x\cdot 1$	
0.0000	$x\cdot 0$	
0.10001111	xy	143/256.

Note that a shift takes place after every addition except the last. Although the above example is written as if the multiplicand were shifted to the left, obviously this is equivalent to shifting the partial product to the right; the latter process is better adapted to computer operation.

Now suppose that the multiplicand, x, is positive and the multiplier, y, is negative. As usual, y is written $2-|y|=2+y$. Multiplying gives $x(2+y)=2x+xy$, which can be corrected to equal the product xy by subtracting $2x$. Although $2x$ is obtainable by a simple shift, nevertheless it would be more convenient if this shift could be avoided, and this turns out to be easily possible. Let x be multiplied by all digits of y except the sign digit. Since y is negative, this sign digit is 1; the result, therefore, is to multiply x by one less than before. This means the formation of the product $x(1+y)=x+xy$ instead of $x(2+y)=2x+xy$. Now, a correction of $-x$ is sufficient, and furthermore, the process is more systematic, since every addition is followed by a shift.

Thus, to take care of the cases of x positive and y either positive or negative, it is only necessary to follow the procedure outlined for both x and y positive, and, if y is negative, correct the result by subtracting x from it. The following example illustrates the procedure:

0.1101	x	$\frac{13}{16}$
1.0101	y	$-\frac{11}{16}$
0.1101		
0.0000		
0.1101		
0.0000		
0.010000001	$xy+x$	
1.0011	$-x$	
1.011100001	xy	$-143/256.$

From symmetry it is apparent that if x is negative and y positive, a correction of $-y$ will give the true product. As a preliminary assumption, let the sign digit of x be ignored until the correction at the end is made. This leaves only the question of how to add in the $-y$. It turns out to be convenient, as far as computer design is concerned, to examine successive digits of y by shifting y to the right a digit at a time, dropping off each digit after it has been examined. This means that by the end of the process y has been lost and is not available for use in correcting the product. Therefore, the possibility of making the correction digit by digit as the product is built up must be examined.

Suppose the numbers being used have 30 digits and a sign. Referring back to the basic multiplication process, it is noted that 30 shifts are made in building up the product; therefore, the sign digit of x during the first addition finally becomes the 30th or least significant digit of the product. Similarly, the sign digit of x on the second addition becomes the 29th digit of the product, and so on. It therefore becomes apparent that the sign digit of x during any addition occupies in the final product a position equivalent to that of the digit of y which controlled its addition into the product.

Next, note that the correction needed is the complement of y. Thus, for every 1 in y a 0 is needed in the correction term, but for every 0 in y a 1 should appear in the correction term. As usual, the complement must be corrected by adding a unit in the last place. Note further that the correction is needed only if x is negative —that is, if its sign digit is 1.

Putting together the above facts, the required procedure may be defined. If a digit of y is 1, x (without its sign digit) is added into the partial product. If a digit of y is 0, its complement, 1, must be added into the partial product, but a 1 is available, in the correct position, in the form of the sign digit of x. Thus, if a digit of y is 0, the sign digit only of x is added. Of course, this sign digit might be 0, but that would imply that x was positive and no correction was required; thus no harm is done by adding the zero. Since the additions are terminated before examining the sign digit of y, the correction term, if any, still lacks a 1 in the sign position and a 1 in the units position. Hence, if x is

negative, these digits must be added to the product, as shown in the following:

```
  1.0 0 1 1   x                    -13/16
  0.1 0 1 1   y                     11/16
  ---------
  0.0 0 1 1   (x without sign)
    0.0 0 1 1
  1 0 0 0 0   (sign only of x)
  0.0 0 1 1
-----------
0 0 1 1 0 0 0 0 1
1 0 0 0 1     (correction term)
-----------
1.0 1 1 1 0 0 0 1   xy             -143/256.
```

Taking the third term of the partial product (that is, 1.0000 shifted twice to become 0.0100) and the correction term, and adding, the result $0.0100+1.0001=1.0101$ is obtained. Thus, it is shown that the complement of y has in fact been added to the product.

Finally, the case of x and y both negative will be considered. Starting with the numerical example, and following the procedure outlined for the preceding cases:

```
  1.0 0 1 1   x                    -13/16
  1.0 1 0 1   y                    -11/16
  ---------
  0.0 0 1 1   (x without sign)
  1.0 0 0 0   (sign only of x)
  0.0 0 1 1
1.0 0 0 0
-----------
0 1 0 1 0 1 1 1 1
1.0 0 0 1     (correction for sign of x)
-----------
1 1 0 1 1 1 1 1 1   xy+x
0 1 1 0 1           -x (correction for sign of y)
-----------
0.1 0 0 0 1 1 1 1   xy              143/256.
```

The correction terms added on account of the sign of x are (1) 1.0000, shifted three times to become 0.0010; (2) 1.0000, shifted once to become 0.1000; (3) 1.0001, added after all shifts are completed.

Again adding these various corrections, the result is 1.1011, which appears to be wrong, because the complement of y is 0.1011. Recalling that, up to the correction steps, signs were disregarded, and that therefore the factors behaved like $x+1$ and $y+1$, the uncorrected product obtained must be $(x+1)(y+1)=xy+x+y+1$. Hence, it is necessary to subtract 1 as well as x and y to get the true product. Since all carryovers beyond the sign are disregarded, subtracting 1 is the same as adding 1. Thus, the extra 1 which appeared in the complement of y is necessary.

The foregoing discussion covers all possible cases and shows that a single generalized multiplication process can be used to produce results which are correct regardless of the signs of the factors.

V. DIVISION

The division process used in the binary computer is the non-restoring method, analogous to that used in desk computing machines. This method has two important advantages—it eliminates time-consuming comparison operations which are necessary in the restoring method, and it requires no extra correction operations if either dividend or divisor is negative.

In the non-restoring method of division, the signs of the dividend and divisor are first examined. If they are alike, the divisor is subtracted from the dividend repeatedly until the signs become different, a unit being added into the corresponding quotient digit for each subtraction. If the signs are different, the divisor is added to the dividend repeatedly until the signs become the same, a unit being subtracted from the corresponding quotient digit each time. Thus, in the decimal system, if the signs were alike and the divisor had to be subtracted three times to cause a sign reversal, the corresponding quotient digit would be 3; if the signs were different and the divisor had to be added seven times to cause a sign reversal, the corresponding quotient digit would be -7. As soon as a sign reversal occurs, the remainder is shifted one place to the left and computation of the next quotient digit begins.

If the above process is carried out exactly as described, any quotient digit may be $\pm n$, where n is an integer in the range 1 to 10 inclusive. It is obvious, however, that if a sign reversal has not occurred after nine additions or subtractions (in the decimal system), it must surely occur on the tenth operation.[3] Thus, it is sufficient to terminate the computation of each digit of the quotient after nine operations and shift the remainder without waiting for a sign reversal. The quotient digits then may be $\pm 1, \pm 2, \cdots, \pm 9$. Suppose the quotient obtained is the sequence $1, -3, 9, 5, -4, 6$. To obtain the quotient in usable form, the aggregate of the negative digits must be subtracted from the aggregate of the positive digits: $109{,}506-30{,}040=79{,}466$. If the quotient was built up in an accumulator (i.e., a register containing an adder so arranged that after entry of a new quantity the register contains the sum of the latter and the previous contents), this process was carried out stepwise as the quotient was built up.

An accumulator must be used to hold the dividend (which becomes the remainder, of course, after the process has started), in order to provide means for adding in the divisor or its complement. It is more convenient to store the quotient digits in a simple register, which cannot add or subtract, thus avoiding the necessity of providing an additional accumulator. Returning for a moment to the decimal example, it is noted that there are 18 possible quotient digit values

[3] An exception to this rule occurs in the case of an improper division, i.e., one in which the dividend is larger than the divisor. In this case, no sign reversal would occur on the tenth operation in the computation of the first quotient digit; it may, in fact, be desirable to perform the tenth operation and institute an automatic corrective procedure if no sign reversal occurs.

$(\pm 1, \pm 2, \cdots, \pm 9)$. By the same reasoning, it follows that in the binary system there are only two values (± 1) and that only one operation (addition or subtraction, as the case may be) is necessary for each quotient digit. Thus, as in multiplication, additions (or subtractions) can alternate with shifts, and the result is a process easy to mechanize. There is still the difficulty, however, that the quotient register has no way of distinguishing -1 from $+1$. An investigation of the possibility of entering a 0 in the quotient wherever a -1 is called for immediately suggests itself, in order to see if the pseudo-quotient so obtained bears some simple relationship to the true quotient, x/y, of two binary quantities. If such a simple relationship exists, it will be reasonable to compute the pseudo-quotient and make any necessary corrections to the latter to obtain the true quotient.

Recalling that -1 should be entered in the quotient if y is added to the remainder, and $+1$ if y is subtracted, 0 is used for each addition and $+1$ for each subtraction. An expression can now be written for the new remainder, r_k, in terms of the old remainder r_{k-1}, the quotient digit p_k, and y:

$$r_k = 2r_{k-1} + (1 - 2p_k)y$$

which reduces to $2r_{k-1}+y$ if p_k is 0 and $2r_{k-1}-y$ if p_k is 1. The factor of 2 results from the fact that the old remainder was shifted to the left before the addition or subtraction of y. Multiplying through by 2^{-k}

$$2^{-k}r_k = 2^{-(k-1)}r_{k-1} + (2^{-k} - 2^{-(k-1)}p_k)y$$

gives a more convenient recursion formula. Noting that the "zeroth" remainder is actually x itself,

$$\begin{aligned} 2^{-1}r_1 &= x + (2^{-1} - 2^0 p_1)y \\ 2^{-2}r_2 &= 2^{-1}r_1 + (2^{-2} - 2^{-1}p_2)y \\ &= x + [(2^{-1} + 2^{-2}) - (2^0 p_1 + 2^{-1}p_2)]y \quad \text{etc.,} \end{aligned}$$

or, in general,

$$2^{-n}r_n = x + \left(\sum_1^n 2^{-k} - \sum_1^n 2^{-(k-1)}p_k\right)y.$$

Now

$$\sum_1^n 2^{-k} = 0.1 + 0.01 + \cdots = 0.111\cdots$$

which is evidently equal to $1 - 2^{-n}$. Thus, making this substitution and transposing,

$$x = \left[-1 + 2^{-n} + \sum_1^n 2^{-(k-1)}p_k\right]y + 2^{-n}r_n$$

and, dividing by y,

$$x/y = \left[-1 + 2^{-n} + \sum_1^n 2^{-(k-1)}p_k\right] + 2^{-n}r_n/y.$$

The expression in square brackets, then, is the desired quotient. Now if $n = 30$, the first 30 quotient digits computed by the method outlined above correspond to the sign and the first 29 digits of the pseudo-quotient. The first digit, p_1, corresponds to the sign rather than the most significant digit because it is in the 2^0 position; $(2^{-(k-1)} = 2^0$ for $k = 1.)$ To convert the pseudo-quotient into the true quotient, it is necessary to add 2^{-30} and subtract one. The first of these corrections corresponds to putting a unit in the least significant digit position, a step which also coincides with the round-off procedure to be adopted. As for subtracting 1 from the result, since carries beyond the sign position are disregarded, subtracting 1 is the same as adding 1; the two corrections can therefore be taken care of by bringing the pseudo-quotient from the register where it was built up into the accumulator, simultaneously clearing out the last remainder and adding units into the sign and the least significant digits. (It may be noted that the latter position is still unfilled, since p_{31} was not computed; the correction in this position could therefore have been made in the register, but an adder is required for the sign correction.)

The reader may have been troubled by the fact that in doubling the remainder the sign digit will be lost. Actually, this does no harm. The addition or subtraction of the divisor is always such as to decrease the absolute value of the remainder; since $x < y$ to begin with, $2x - y < y$ and in general $2r - y < y$. Since $y < 1$, $2r - y < 1$, and therefore the result of the operation is always within the range of the machine; although the sign digit of r was shifted out of range, it could not have affected the result in any case. It should be noted that the sign of r is used to determine whether y is to be added or subtracted, however; therefore, since this sign is lost by the shift, it must be examined and compared with the sign of y before the shift takes place.

The example below illustrates the division process just described:

1.0111	$x(r_0)$	$-\frac{9}{16}$
0.1011	y	$\frac{11}{16}$
0.1110	$2r_0$	
0.1011	$+y$	
1.1001	r_1	$(p_1 = 0)$
1.0010	$2r_1$	
0.1011	$+y$	
1.1101	r_2	$(p_2 = 0)$
1.1010	$2r_2$	
0.1011	$+y$	
0.0101	r_3	$(p_3 = 0)$
0.1010	$2r_3$	
1.0101	$-y$	
1.1111	r_4	$(p_4 = 1)$
0.0010	$x/y + 1 - 2^{-4}$	
1.0001	$1 + 2^{-4}$	
1.0011	$x/y + 2 = x/y$	$-\frac{13}{16}$

TABLE III.

Minuend	Subtrahend	Difference	Borrow
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

which is the exact quotient, $-9/11$, rounded off to four binary places. r_4 is discarded; the pseudo-quotient ($p_1+p_2+p_3+p_4=0.001$) and the correction term 1.0001 are added together and the result left in the accumulator.

VI. ROUND-OFF PROCEDURES

Both multiplication and division give results having more significant figures than the original numbers entering into these processes; in general, multiplication of two n-digit factors yields a product of $2n$ digits, and division yields a quotient having an infinite number of digits. Since only n digits are required for most computational processes, some round-off procedure must be followed in order to avoid the bias which would result if the remaining digits were simply dropped. The round-off procedure should fulfill two requirements: the approximation should lie as close to the true product or quotient as practical, and it should be unbiased, i.e., its mean should equal the true result.

Of the various possible round-off schemes, two are of interest here. The first and most familiar consists of adding a unit to the $n+1$st digit, allowing any resulting carries to take place, and then keeping only the first n digits. The other consists of replacing the nth digit with a 1, regardless of whether that digit was originally a 1 or a 0. It can be shown that both schemes produce results which are sufficiently unbiased in view of the fact that products and quotients themselves are not entirely unbiased.[4] The latter situation is due to the fact that the original quantities themselves are, in general, rounded-off approximations of other quantities. On the other hand, the variance of the second method is twice as great as that of the first, being 0.58 times the last digit as compared to 0.29 times the last digit for the more familiar scheme.

In spite of the larger variance, however, the second round-off scheme is used. It has the advantage of being usable even after the $n+1$st digit has been lost as the result of a right shift; thus, it is unnecessary to interrupt the multiplication process before the last step to add in the unit for round-off. Furthermore, no carries are produced; thus, the unit pulse can be put in after the adder, and the process carried out concurrently with the last complement correction. In the case of division, as has already been seen, the result obtained already conforms to this round-off procedure without further modifications; it therefore becomes unnecessary to compute the 31st quotient digit.

[4] See reference 1, p. 20.

TABLE IV.

M	S	B	D	B
0	0	0	0	0
1	0	0	1	0
1	1	0	0	0
1	0	1	0	0
1	1	1	1	1
0	1	0	1	1
0	0	1	1	1
0	1	1	0	1

When it is necessary during computation to round off to less than 30 digits, the more precise method can be used; in most cases the procedure will be to shift right until all but one of the unwanted digits are lost, add a unit in the lowest position, and then shift right one more place.

VII. STORAGE OF ABSOLUTE VALUES

While negative quantities are stored in complement form in the Binac and most other binary computers now under construction, it is of interest to examine briefly the possibility of storing the absolute values of negative quantities, distinguishing them from positive quantities only by the difference in the sign digit; in this case it is immaterial whether 0 and 1 are used for + and −, respectively, or vice versa. There are two principal advantages to such a procedure; first, the preparation of input data and the interpretation of output data are simplified; and second, the visual interpretation of data in registers and memory is made easier, thus expediting trouble-shooting.

The effect on multiplication and division is that of simplification, since in these operations the signs can be disregarded and the operations performed on absolute values with no corrections necessary. A preliminary comparison of the signs of the operands discloses immediately the sign of the result; this is stored until the operation is complete and then appended to the product or quotient.

In the case of addition, however, the operations become more complex. Before addition can take place, the signs of the addends must be compared; if they are alike, an addition is performed; otherwise subtraction is necessary, with some provision for assuring the subtraction of the operand having the smaller absolute value from that having the larger, or for making a suitable correction which will produce the same effect. The subtraction can be accomplished in a regular adder by complementing negative quantities before sending them to the adder, but it is possible to avoid complements completely, even in the arithmetic circuits, by using a subtractor rather than an adder when such operation is indicated. The subtractor proves to be as simple electronically as an adder, since it must fulfill a set of conditions differing only slightly from those satisfied by the adder.

For two binary digits, the rules for subtraction can be summarized as in Table III. To complete the subtraction, provision must be made for taking the "borrow" pulse and subtracting it from the next minuend digit. This is equivalent, however, to adding it to the next subtrahend digit; and the latter procedure proves to be more advantageous electronically. The full set of conditions can now be written down as in Table IV.

The preceding discussion of the storage of absolute values has been included here partly to illustrate the difficulties to be encountered in such a system, and partly because of the importance of the system in computers of the decimal type, even though the latter are not otherwise considered in this paper. Because of the complications involved, storage of absolute values is not ordinarily employed in binary computers, probably because it is felt that the inconvenience of representing quantities in binary form is not materially increased by the necessity of representing negative quantities in complement form. In decimal computers such as the Univac, however, where convenience in handling of input and output data is of great importance, the storage of complements is avoided.

THE REVIEW OF SCIENTIFIC INSTRUMENTS VOLUME 21, NUMBER 8 AUGUST, 1950

A Recording Polarimeter

GABOR B. LEVY, PHILIP SCHWED,* AND DAVID FERGUS
Research Department, Schenley Laboratories, Inc., Lawrenceburg, Indiana

(Received March 23, 1950)

A recording polarimeter has been constructed. The principle of operation is essentially the replacement of an operator in conventional visual polarimetry by a photoelectric servo system. A photo-multiplier tube is used which, in conjunction with a mechanical chopping system, delivers an alternating current signal proportional to the unbalance. The latter is filtered and amplified and actuates a servo mechanism. The precision of the system is $\pm 0.005°$ for static balance and the speed about 0.5° per minute. Hence the arrangement compares favorably with ordinary visual operation. The construction of the instrument is discussed briefly and its performance is illustrated by examples.

INTRODUCTION

IN the study of chemical or biochemical reactions, it is of great advantage to select a method of analysis that is non-destructive and which does not involve sampling. Potentially, polarimetric measurements are suitable, particularly in the biochemical field, since the optical activity is frequently subject to change in the course of the reaction. However, the measurement of changing optical rotation by visual methods is tedious and of limited precision. For such applications, a recording polarimeter would be of great value.

Instruments of this type are not available, and furthermore, the existing literature gives no encouragement for success of such a scheme. For example, W. Heller[1] states: "Persons with particularly sensitive, specially trained, and well-adapted eyes can be expected, however, to surpass any photoelectric registration. . . ."

Considering the fact that measurement of fast changing rotation is far from an ideal condition for visual observation, we felt encouraged to construct a photoelectric instrument despite the statement cited. In addition, the potential value of such an instrument appeared to warrant the effort.

We have chosen the most direct approach, i.e., converting a visual instrument to photoelectric operation. The performance of the resultant instrument and its potential value in the study of optical activity seems to justify its description in some detail.

PRINCIPLES OF OPERATION

Essentially monochromatic light is passed through a conventional polarimeter. The "half-shade" image from its Lippich analyzer (with two fields) is projected onto a photo-multiplier photo-cell. Interposed is a semicircular sector which is driven by a synchronous motor. Thus the output of the photo-cell, at unbalance, is a sinusoidal alternating current whose phase is shifted by 180° when the direction of the unbalance reverses. This signal is amplified by a battery operated untuned preamplifier and further amplified by an amplifier tuned to the fixed frequency of the sector and signal. The signal, which is thus freed from noise, is fed after further amplification to one of the two coils of a differential relay, the coil energized depending on the position of the sector. This arrangement provides phase discrimination and rectification as well as further filtering. The relay actuates a reversible 60-cycle motor which is mechanically linked to the analyzer of the

* Now with the National Advisory Committee for Aeronautics, Cleveland, Ohio.

[1] W. Heller, in A. Weissberger, *Physical Methods of Organic Chemistry*, Vol. II (Interscience Publishers, Inc., New York, N. Y., 1946), p. 967.