Correction to "LOGICAL" ARITHMETIC ON COMPUTERS WITH TWO'S COMPLEMENT BINARY ARITHMETIC

Communications of the ACM, Volume 11, Number 7 (July 1968) 517.

In reference (1) algorithms for performing arithmetic with unsigned two's complement operands were described. The scheme for division was implemented in a set of multiple-precision floating-point arithmetic routines (2). User experience with those routines showed that there is one case where the algorithm fails (3). We will give here a modification to the algorithm which eliminates the error condition. Equations will be numbered beginning with (2), so that we may refer to equations in the original paper as well.

Using the notation of (1), it may happen when B = 1 that

$$\left[\frac{4X + A}{M}\right] = 2Y \ . \tag{20}$$

That is, after the low-order bit of the divisor has been dropped, it is possible that the divisor Y is now equal to the high-order part of the dividend, whereas it was strictly greater than the high-order part of the dividend when the bit was present. This situation leads to a fixed point division error in the calculation of the quotient Q, since we now expect to find that $Q = M/2$ (which will be shown below). We will see that the algorithm shown in Figure 2 and programmed in Figure 3 of reference (1) can be modified simply to handle this case.

First, observe that we may rewrite equation (20) in the form

$$4X = 2MY + G \tag{21}$$

where G satisfies the inequalities

$$0 \leq G \leq M-4 \ . \tag{22}$$

(We have used the definition of A in equation (10a) to eliminate the two low-order bits of the dividend; otherwise the upper bound on G would be M-1.) If we insert these two relations into equation (16), we find that

$$M - 2 \leq QUOT \leq M - 1 \ . \tag{23}$$

This gives a bound on the size of the true quotient QUOT. To verify that the trial quotient Q is indeed M/2, we can insert equations (21) and (22) into equation (15) to find that

$$\left(\frac{M}{2} - 1\right) + \frac{4}{M} \leq Q \leq \left(\frac{M}{2} + 1\right) - \frac{4}{M} \quad . \qquad (24)$$

Because Q must be an integer, we have $M/2 \leq Q \leq M/2$, as desired. Note that the relationship (19) between the trial and true quotients is still satisfied; in particular, at least one correction to the trial quotient is always required.

That these bounds are achieved may be seen by considering the following examples.

Dividend	Divisor	QUOT	REM	Q	R
M^2-M-1	M-1	M-1	M-2	$\frac{1}{2}$M	$\frac{1}{4}$M-1
M^2-2M	M-1	M-2	M-2	$\frac{1}{2}$M	0

To see which parts of the algorithm need to be modified, we can insert equation (21) into equation (14); we find that most of the terms cancel, leaving G = 4R. Since G satisfies equation (22), we find immediately that

$$0 < R < \frac{M}{4} - 1 \quad . \qquad (25)$$

This means that in forming the quantity (4R + A), we cannot have an overflow in this special case; hence the correction of the tentative value of QUOT (namely, M) is very simple, and is shown in Figure 4 below. The program segment of Figure 3 is corrected in Figure 5 below.

There is one other minor correction to reference (1); in the second sentence of section 5, the word "positive" should read "negative".

References

1. "Logical Arithmetic on Computers with Two's Complement Binary Arithmetic", Communications of the ACM, Volume 11, Number 7 (1968), page 517.
2. "Multiple-Precision Floating-Point Arithmetic Package", available from IBM's Program Information Department as Program Number 360D-40.4.003.
3. Private Communication from Hirondo Kuki, University of Chicago.

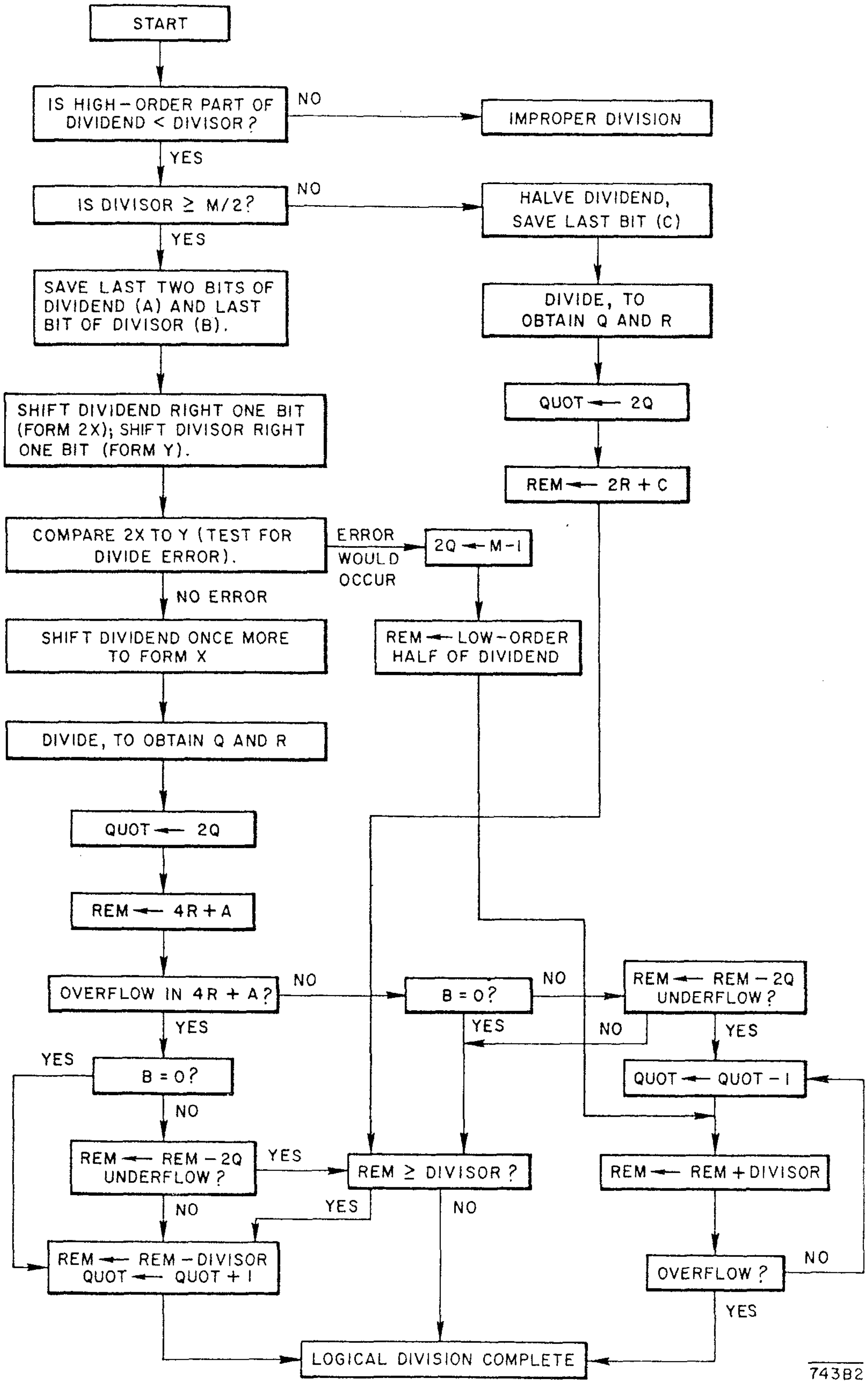

FIG. 4 --LOGICAL DIVISION (CORRECTED)

```
         LM    0,1,DIVIDEND        GET DIVIDEND IN R0, R1
         CL    0,DIVISOR           CHECK FOR INVALID DIVISION
         BC    10,ERROR1           BRANCH IF IMPOSSIBLE
         TM    DIVISOR,X'80'       SEE IF DIVISOR SIGN BIT IS 1
         BC    1,P                 JUMP IF YES FOR HARD CASES
         SRDL  0,1                 SHIFT DIVIDEND RIGHT 1 BIT
         D     0,DIVISOR           DIVIDE BY POSITIVE DIVISOR
         SLDL  0,1                 DOUBLE QUOTIENT AND REMAINDER
         TM    DIVIDEND+7,1        SEE IF LAST DIVIDEND BIT WAS 1
         BC    8,X                 BRANCH IF NOT TO CORRECT
         AL    0,=F'1'             OTHERWISE RESTORE IT IN REMAINDER
         B     X                   AND GO COMPLETE THE DIVISION
*
P        LTR   0,0                 CHECK FOR HIGH PART OF DIVIDEND = 0
         BC    7,A                 JUMP IF NOT, NO FURTHER SIMPLE CASES
         LR    0,1                 OTHERWISE SET UP TO SKIP DIVISION
         SR    1,1                 SET TENTATIVE QUOTIENT TO ZERO
         B     X                   AND GO FINISH UP CORRECTLY
*
A        LA    2,3                 MASK BITS FOR 'A' IN REGISTER 2
         NR    2,1                 LOGICAL 'AND' SAVES THE 2 BITS
         SRDL  0,1                 SHIFT RIGHT ONE POSITION FOR TEST
         L     3,DIVISOR           GET DIVISOR FOR TEST AND DIVISION
         SRL   3,1                 DIVIDE DIVISOR BY 2 (FORM 'Y')
         CLR   0,3                 COMPARE (2X/M) TO 'Y'
         BC    4,D                 BRANCH IF SMALLER, DIVISION PROCEEDS
         L     0,DIVIDEND+4        SET (4R+A) FROM LOW-ORDER DIVIDEND
         SR    1,1                 SET QUOT TO M (WHICH IS THE SAME AS 0)
         B     C                   AND ENTER CORRECTION SEQUENCE
D        SRDL  0,1                 COMPLETE THE POSITIONING OF 'X'
         DR    0,3                 DIVIDE, R AND Q IN REGISTERS 0 AND 1
         SLDL  0,1                 2R AND 2Q
         ALR   0,0                 FORM 4R IN REGISTER 0
         BC    3,B                 BRANCH IF 4R OVERFLOWS THE REGISTER
         ALR   0,2                 REGISTER 0 HAS 4R+A, NO OVERFLOW
         TM    DIVISOR+3,1         TEST IF 'B' WAS 1
         BC    8,X                 JUMP IF B = 0, ONLY ONE CORRECTION
         SLR   0,1                 OTHERWISE FORM 4R+A-2Q IN REGISTER 0
         BC    3,X                 JUMP IF NO UNDERFLOW, RESULT IN RANGE
C        SL    1,=F'1'             OTHERWISE, QUOT = 2Q - 1, AND...
         AL    0,DIVISOR           ...REM = 4R+A - 2Q + DIVISOR.
         BC    12,C                JUMP BACK IF ONE MORE CORRECTION
         B     OUT                 EXIT
*
B        ALR   0,2                 4R+A, WITH OVERFLOW IMPLIED
         TM    DIVISOR+3,1         TEST IF 'B' WAS 1
         BC    8,Y                 BRANCH IF NOT
         SLR   0,1                 FORM 4R+A - 2Q
         BC    3,Y                 IF NO UNDERFLOW, OVERFLOW STILL IMPLIED
*
X        CL    0,DIVISOR           SEE IF REMAINDER IS LESS THAN DIVISOR
         BC    4,OUT               IF SO, WE'RE FINISHED, EXIT.
Y        SL    0,DIVISOR           OTHERWISE CORRECT THE REMAINDER
         AL    1,=F'1'             AND INCREMENT THE QUOTIENT BY 1
*
OUT      ST    0,REMAINDR          STORE FINAL REMAINDER
         ST    1,QUOTIENT          AND FINAL QUOTIENT.
```

FIG. 5--Code Sequence (Revised).

SLAC-PUB-302
April 1967

"Logical" Arithmetic on Computers with Two's Complement Binary Arithmetic

By: John R. Ehrman

Stanford Linear Accelerator Center
Stanford University
Stanford, California

It is often useful to be able to treat all the digits of a signed word in a binary computer as having positive weight: for example, an additional factor of two in the allowed range of some numbers may be sufficient to allow the solution of certain problems not otherwise easily handled; and in the coding of multiple precision arithmetic, such numbers often arise in a natural way. It was in this latter context [1] that the author found it necessary to devise the methods described below for the multiplication and division of numbers in such a "logical" representation.

I. Two's Complement Representation

Suppose we are working with a machine with registers of length N binary digits, and take $M = 2^N$. Then the logical representation of a whole number X requires that X satisfy

$$0 \leq X \leq M - 1. \tag{1}$$

The two's complement, or arithmetic, representation of a number x which lies in the range $0 \leq x \leq \frac{1}{2}M - 1$ is

$$x = X, \quad (x \geq 0) \tag{2}$$

and the representation of negative number x which lies in the range $-\frac{1}{2}M \leq x \leq -1$ is

$$x = X - M. \quad (x < 0) \tag{3}$$

All numbers in the following discussion will be assumed to be integer; the results of any operation may of course be considered as fractions by including the appropriate scale factors.

(Submitted to ACM)

II. Addition and Subtraction

In a computer with two's complement arithmetic, the sign bit of a number is treated during addition and subtraction modulo M as an ordinary numeric digit, so that no adjustments need be made to the result, other than to note the presence or absence of a carry out of the leftmost digit position. It is useful to remember that when performing a logical subtraction, a carry will occur if the result is "in range", that is, if the logical minuend is not smaller than the logical subtrahend; in simpler terms, this means that the result has not "gone negative" in an arithmetic sense. In the discussion which follows, a carry out of the most significant digit position during addition will be called an overflow, and the lack of a carry out of the most significant digit position during subtraction will be called an underflow.

III. Multiplication

The multiply instruction on most computers yields an arithmetic product: that is, the multiplier and multiplicand are treated as signed operands. If a logical product is required, some adjustments to the product as computed may be required.

Let X and Y be two logical integers and let x and y be their corresponding arithmetic values. Then if $x*y$ denotes the machine operation of multiplication of the two arithmetic operands x and y, and $X \times Y$ denotes the logical product of X and Y,

a) if $x \geq 0,\ y \geq 0,$

$$X \times Y = x*y; \tag{4}$$

b) if $x < 0,\ y \geq 0,$

$$X \times Y = (M+x)*y = Mx+x*y \ (\text{modulo } M^2); \tag{5}$$

c) if $x \geq 0,\ y < 0,$

$$X \times Y = x*(M+y) = My+x*y \ (\text{modulo } M^2); \tag{6}$$

d) if $x < 0,\ y < 0,$

$$X \times Y = (M+x)*(M+y) = Mx+My+x*y \ (\text{modulo } M^2). \tag{7}$$

Since the product $x*y$ is developed in a double-length register pair of 2N bits, the logical product is formed simply by adding the appropriate terms to the high-order register of the pair, as indicated in the flow diagram in Figure 1.

It should be noted from equations (5) and (6) that if one of the operands is known always to have a non-negative arithmetic representation, (for example, it is known that the multiplier P always satisfies $0 \leq P \leq \frac{1}{2}M - 1$), then the product itself may be tested for sign: if it is negative, add P to the high-order register, and the logical product is complete.

IV. Division

The problem of logical division is more complicated, because the relationship (3) between the logical and arithmetic representations cannot be exploited as in the case of multiplication. A quotient of N binary digits must be formed, whereas the usual machine operation of division produces a quotient of N-1 digits plus sign.

We will suppose that we are given a double-length logical DIVIDEND and a single-length logical DIVISOR, and wish to find the logical quotient QUOT and logical remainder REM which satisfy

$$\text{DIVIDEND} = (\text{QUOT})\times(\text{DIVISOR}) + \text{REM},$$
$$0 \leq \text{REM} \leq \text{DIVISOR} - 1. \qquad (8)$$

If it is known that DIVISOR $\leq \frac{1}{2}M-1$, it always has a positive arithmetic representation, and a simple scheme may be used to perform the division.

(a) Divide the dividend by 2 by performing a logical right shift of one bit position; remember the value of the bit which is shifted off.

(b) Perform the normal machine operation of integer division of the positive dividend by the positive divisor.

(c) Double the resulting quotient and remainder; if the lowest-order dividend bit remembered in step (a) was a one, add one to the doubled remainder.

(d) If the new remainder is logically greater than or equal to the divisor, subtract the divisor from it to give the true remainder and add a low-order one to the doubled quotient to give the true quotient.

This yields a quotient which may occupy a full N bits, and is therefore the analogue of the case in multiplication in which one operand is known always to have a non-negative arithmetic representation.

If the divisor has a negative arithmetic representation, a more complicated scheme must be used. The method used will be described in some detail.

Let DIVIDEND = $4X+A$, where

$$0 \leq A \leq 3. \tag{9a}$$

Similarly, let DIVISOR = $2Y+B$, where

$$0 \leq B \leq 1. \tag{9b}$$

Thus A is the two low-order bits of the dividend, and B is the low-order bit of the divisor. We will assume that $\frac{1}{2}M \leq$ DIVISOR $\leq M-1$.

Since the largest possible value for QUOT is M-1, it is clear that the dividend must satisfy the inequality

$$\begin{aligned} 4X+A &\leq (2Y+B)\times(M-1)+(2Y+B-1) \\ &\leq M(2Y+B) - 1. \end{aligned} \tag{10}$$

which can also be written

$$\left[\frac{4X+A}{M}\right] < \quad 2Y+B,$$

where the square brackets mean that [Z] is the largest integer contained in Z. Thus the division will be improper if the register containing the high-order half of the dividend is not logically smaller than the divisor.

To find QUOT and REM, first compute Q and R from

$$X = QY+R \text{ , where} \tag{11}$$

$$0 \leq R \leq Y-1. \tag{12}$$

Then

$$4X+A = (2Y+B)(2Q) + (4R+A-2BQ). \tag{13}$$

By examining the final term in parentheses in equation (13) it is possible to make the necessary corrections and obtain the true values of QUOT and REM. To determine the corrections needed, we will examine the difference between the tentative quotient 2Q and the true quotient QUOT.

From equations (11) and (12) we find

$$\frac{X}{Y} - \frac{Y-1}{Y} \leq Q \leq \frac{X}{Y} , \tag{14}$$

and from equations (8) and (9),

$$\frac{4X+A}{2Y+B} - \frac{2Y+B-1}{2Y+B} \leq \text{QUOT} \leq \frac{4X+A}{2Y+B} . \tag{15}$$

Combining these, we have after some algebra that

$$-2+ \frac{2BX+2+Y(4-A)}{Y(2Y+B)} \leq 2Q\text{-QUOT} \leq 1 + \frac{2XB-Y(A+1)}{Y(2Y+B)} . \tag{16}$$

Case I: B=0.

It can be seen that the bounds on the difference 2Q-QUOT are most restrictive when A=0 and Y takes on its minimum value, which by assumption is M/4, since DIVISOR $\geq$M/2. It can then be seen that

$$-2 + \frac{8}{M} \leq 2Q\text{-QUOT} \leq 1 - \frac{8}{M} ,$$

and if we are operating in a machine with registers of length greater than three bits (M>8), the fact that 2Q and QUOT are integers allows us to write the inequalities as

$$-1 \leq 2Q\text{-QUOT} \leq 0. \tag{17}$$

Thus at most one correction must be made to the tentative remainder 4R+A.

Case II: B=1.

In this case the bounds depend on the size of X. For the largest possible value of X obtainable from equation (10), it is again found that the bounds are most restrictive when Y=M/4.

This leads to

$$\frac{2(4-A)}{M+2} + \frac{4(3-A)}{M(M+2)} \leq 2Q\text{-QUOT} \leq 3 - \frac{2(A+1)}{M+2} - \frac{4(A+1)}{M(M+2)} ,$$

and since $0 \leq A \leq 3$, this may be reduced to the integer inequalities (again assuming $M > 8$)

$$1 \leq 2Q\text{-QUOT} \leq 2. \tag{18}$$

That the upper bound is actually achieved may be seen by considering the case DIVIDEND = $\frac{1}{2}M^2-M$, DIVISOR = $\frac{1}{2}M+1$.

For the smallest possible value of X (namely zero), it is found that the bounds on the difference 2Q-QUOT are most restrictive when $Y = M/4$, which gives

$$-2 + \frac{2(4-A)}{M+2} + \frac{16}{M(M+2)} \leq 2Q\text{-QUOT} \leq 1 - \frac{2(A+1)}{M+2} ,$$

which on the same assumptions leads to

$$-1 \leq 2Q\text{-QUOT} \leq 0. \tag{19}$$

By considering the full range of values of X, we can combine (18) and (19) to obtain

$$-1 \leq 2Q\text{-QUOT} \leq 2 \tag{20}$$

for the case B=1.

A flow diagram which indicates the overall division process is shown in Figure 2.

V. Sample Program

A program was written for an IBM System/360 (Model 50) which tested the division algorithm given above. Random 32-bit fullword integers were generated for DIVISOR, QUOT, and REM, subject to the restriction REM < DIVISOR. The value for QUOT was then divided by 2^k, where k was an integer chosen randomly in the interval $0 \leq k \leq 31$. The dividend was then computed from equation (8), and the division of DIVIDEND by DIVISOR begun. The resulting quotient and remainder were compared to the known values, and diagnostic information was printed in case of any disagreement. Over 18 million separate tests using several different random number generators were made of the division algorithm at a rate of 100,000/

minute. The bounds on the difference between the true and tentative quotients given in equations (17) and (20) were verified, and the algorithm is known to be correct.

The portion of the program which performs the Logical division is given in Figure 3. It contains one additional test not shown in Figure 2: if the divisor has a negative arithmetic representation, and DIVIDEND<M (that is, the high-order part of the dividend is zero) then the division process may be skipped. Because all System/360 fixed-point addition instructions (as well as the logical OR instruction) change the condition code [2], the quantity 4R+A must be computed by first forming 4R and testing it for overflow, and later adding A, which cannot then cause an additional overflow.

VI. References

[1] Stanford Linear Accelerator Center Computation Group Program Library Routine No. A041: "Multiple Precision Floating-Point Arithmetic Subroutine Package"

[2] IBM System/360 Principles of Operation, File No. 5360-01, Form A22-6821, International Business Machines Corporation.

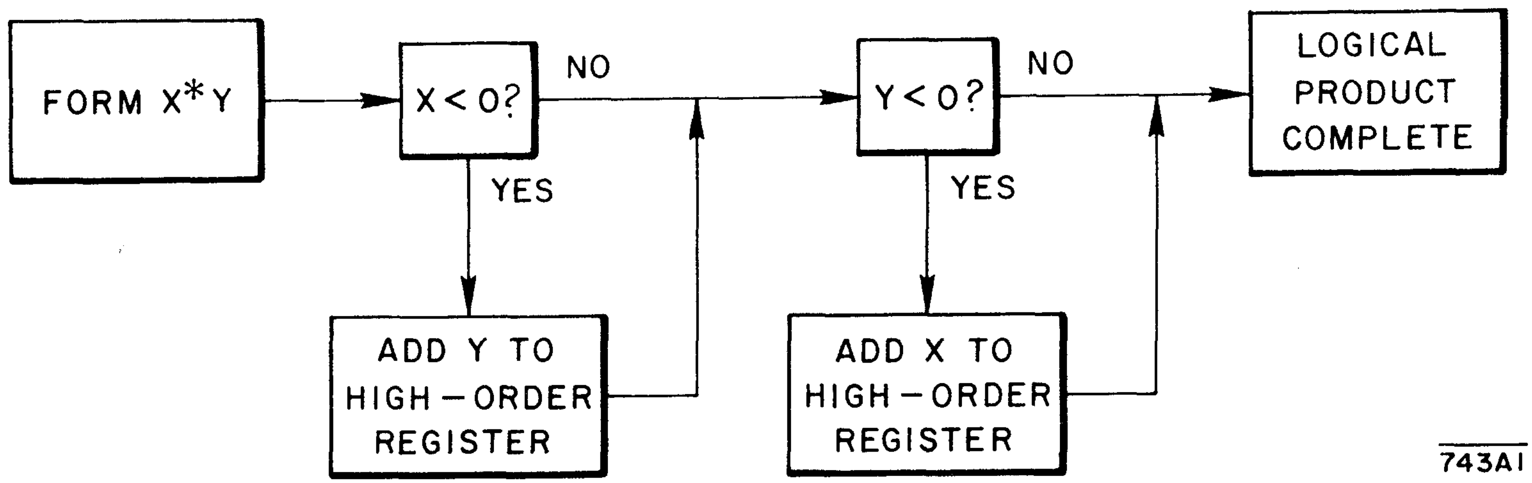

FIG. 1 -- LOGICAL MULTIPLICATION

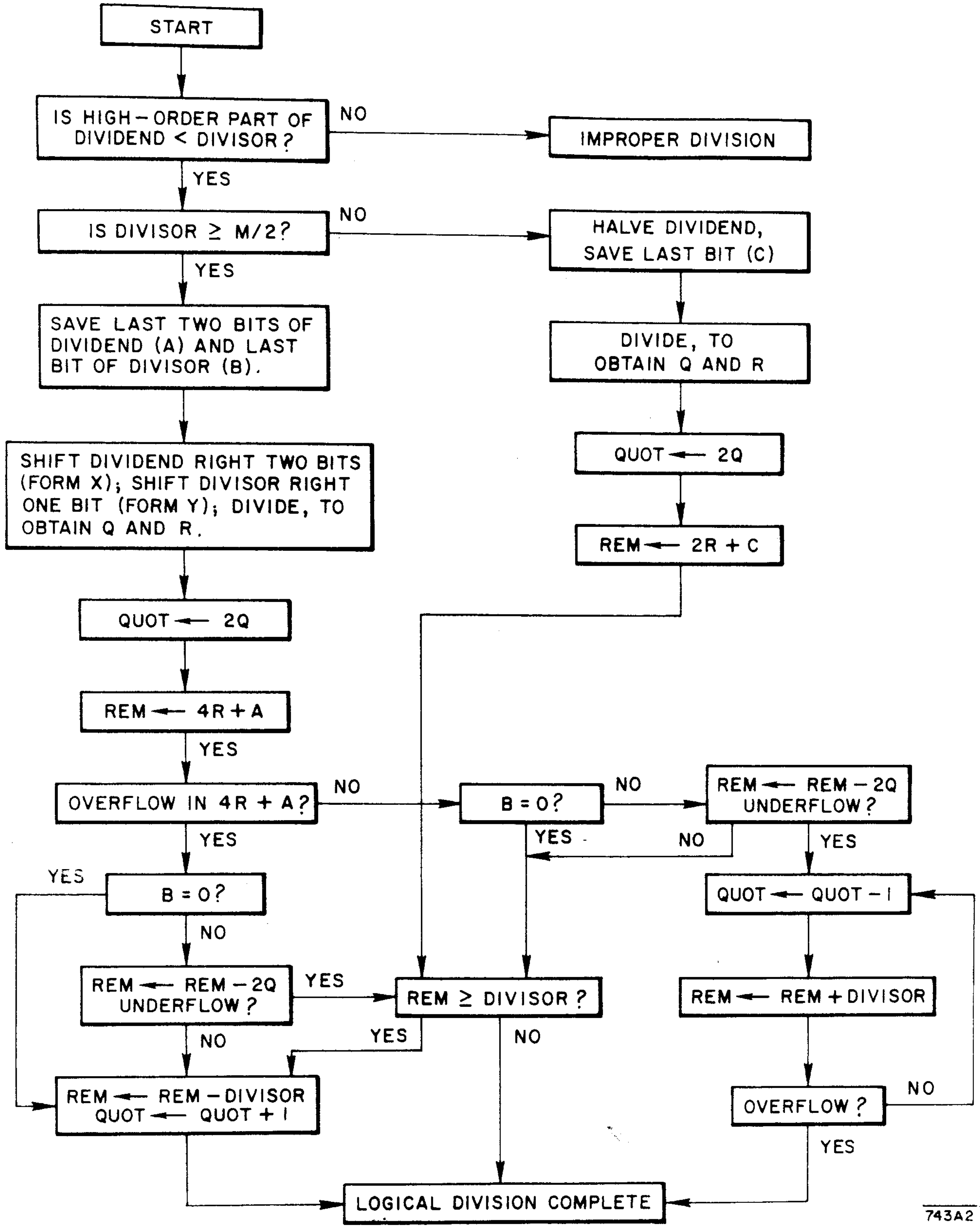

FIG. 2 -- LOGICAL DIVISION

```
      LM     0,1,DIVIDEND
      CL     0,DIVISOR    CHECK FOR INVALID DIVISION
      BC     10,ERROR1    BRANCH IF IMPOSSIBLE
      TM     DIVISOR,X'80'      SEE IF DIVISOR SIGN BIT IS 1
      BC     1,P          JUMP IF YES
      SRDL   0,1          SHIFT DIVIDEND RIGHT 1 BIT
      D      0,DIVISOR    DIVIDE BY POSITIVE DIVISOR
      SLDL   0,1          DOUBLE QUOTIENT AND REMAINDER
      TM     DIVIDEND+7,1       SEE IF LAST BIT OF DIVIDEND WAS 1
      BC     8,X          JUMP IF NOT
      AL     0,=F'1'      OTHERWISE PUT IT BACK IN THE REMAINDER
      B      X            AND GO COMPLETE THE DIVISION
*
P     LTR    0,0          CHECK FOR UPPER HALF OF DIVIDEND = 0
      BC     7,A          JUMP IF NOT
      LR     0,1          OTHERWISE SET UP TO SKIP DIVISION
      SR     1,1          SET TENTATIVE QUOTIENT TO 0
      B      X            AND GO FINISH UP
*
A     LA     2,3          MASK BITS FOR A IN REGISTER 2
      NR     2,1          LOGICAL AND SAVES THE TWO BITS OF A
      SRDL   0,2          X IN REGISTERS 0 AND 1
      L      3,DIVISOR
      SRL    3,1          REGISTER 3 NOW HAS Y
      DR     0,3          DIVIDE, GIVING R AND Q IN REGISTERS 0 AND 1
      SLDL   0,1          2R AND 2Q
      ALR    0,0          4R IN REGISTER 0
      BC     3,B          JUMP IF 4R OVERFLOWS THE REGISTER
      ALR    0,2          REGISTER 0 NOW HAS 4R+A
      TM     DIVISOR+3,1  TEST IF B IS 1
      BC     8,X          JUMP IF B = 0, ONLY ONE CORRECTION NEEDED
      SLR    0,1          OTHERWISE FORM 4R+A-2Q IN REGISTER 0
      BC     3,X          JUMP IF NO UNDERFLOW, IT'S IN RANGE
C     SL     1,=F'1'      OTHERWISE QUOT = 2Q - 1, AND
      AL     0,DIVISOR    REM = 4R+A - 2Q + DIVISOR.
      BC     12,C         JUMP BACK IF ONE MORE CORRECTION NEEDED
      B      OUT          EXIT
*
B     ALR    0,2          4R+A, WITH OVERFLOW IMPLIED
      TM     DIVISOR+3,1  TEST IF B = 1
      BC     8,Y          JUMP IF NOT
      SLR    0,1          4R+A-2Q
      BC     3,Y          IF NO UNDERFLOW, AN OVERFLOW IS STILL IMPLIED
*
X     CL     0,DIVISOR    SEE IF REMAINDER IS LESS THAN DIVISOR
      BC     4,OUT        JUMP IF IT IS, WE'RE DONE
Y     SL     0,DIVISOR    OTHERWISE CORRECT THE REMAINDER
      AL     1,=F'1'      AND THE QUOTIENT
*
OUT   ST     0,REMAINDR   STORE REMAINDER
      ST     1,QUOTIENT   AND QUOTIENT
```

743A3

Fig. 3