MASTER COPY

CGTM # 30
John Ehrman
November 1967

Algorithms for Performing Multiplication and Division with "Logical" Operands

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

by: John R. Ehrman
Stanford Linear Accelerator Center
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: an additional factor of two in the allowed range of some numbers may be sufficient to permit the solution of problems not easily handled otherwise, or the natural representation of certain types of data may take this form. It is usually found that addition and subtraction of such quantities is straightforward, while multiplication and division appear to be difficult. An approach sometimes taken in such situations is to find an indirect way to perform the desired processing, perhaps at the cost of wasting one or more bits per word; however, if the operands find their natural representation to be such that there are no insignificant bits in a word, such schemes can become lengthy and awkward. It is the purpose of this note to describe methods for performing multiplication and division directly with such "logical" quantities on machines with two's complement binary arithmetic, so that no recoding of the data is required. This may permit more efficient programming of algorithms for performing processing such as multiple precision arithmetic [1], binary-to-decimal conversion, or number-theoretic calculations.

In general, the algorithms require only that the machine's computed product of two N-bit operands contain 2N significant bits. On computers where the product contains 2N-1 bits, a simplified form of the algorithms may be used which allows the product of unsigned N-bit and (N-1)-bit operands to be calculated, as well as the calculation of an unsigned N-bit quotient from a (2N-1)-bit dividend and an unsigned (N-1)-bit divisor.

* Supported by U.S. Atomic Energy Commission

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 or unsigned representation of a number X requires that X satisfy

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

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

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

or the representation of a negative number x in the range $-\frac{1}{2}M \leq x \leq -1$ is

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

To summarize the above, we can say that the logical quantity X satisfying (1) corresponding to an arithmetic quantity x satisfying $-\frac{1}{2}M \leq x \leq \frac{1}{2}M-1$ is

$$X = (x + M) \quad (\text{modulo } M). \tag{4}$$

In a computer with two's complement binary arithmetic, the sign bit of a number is treated as an ordinary numeric digit during addition and subtraction. Thus, in manipulating numbers in their logical representation, it is only necessary to note the presence or absence of a carry out of the most significant digit position during addition and subtraction. In the discussions which follow, we will use the term overflow to mean that a carry has occurred during addition, and underflow to mean that no carry has occurred during subtraction: the implication of each is that the treatment of the next higher-order arithmetic term (if any) must include respectively a carry or borrow of 1 in the least significant digit.

II. Multiplication

The multiply instruction on must 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 computed product 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 × Y denotes the logical product of X and Y,

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

$$X \times Y = x*y; \tag{5}$$

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

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

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

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

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

$$X \times Y = (M+x)*(M+y) = Mx + My + x*y \text{ (modulo } M^2). \tag{8}$$

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 can be seen that if one of the operands is known always to have a non-negative arithmetic representation (for example, the multiplier P satisfies $0 \leq P \leq \frac{1}{2}M-1$), then the sign of the product itself can be tested; if it is negative, add P and the logical product is complete. This can lead to occasional simplifications in the code required for forming the product. If the machine's computed product is 2N-1 bits long, this is the only case that can be handled.

III. Division

The problem of logical division is more complicated, because the relationship given in equation (4) between the logical and arithmetic represent-

ations 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

$$\text{DIVIDEND} = (\text{QUOT}) \times (\text{DIVISOR}) + \text{REM},$$
$$0 \leq \text{REM} \leq \text{DIVISOR} - 1. \qquad (9)$$

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 either one operand is known always to have a non-negative arithmetic representation, or the dividend has only 2N-1 significant bits.

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

Let DIVIDEND = 4X+A, where

$$0 \le A \le 3. \tag{10a}$$

Similarly, let DIVISOR = 2Y+B, where

$$0 \le B \le 1. \tag{10b}$$

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 $\frac{1}{2}M \le$ DIVISOR $\le M-1$. Since the largest possible value for QUOT is M-1, it is clear that the dividend must satisfy the inequality

$$\begin{aligned} 4X+A &\le (2Y+B)\times(M-1) + (2Y+B-1) \\ &= M(2Y+B) - 1, \end{aligned} \tag{11}$$

which can also be written

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

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, use the ordinary operation of integer division to compute Q and R from

$$X = QY+R, \tag{12}$$

where

$$0 \le R \le Y-1. \tag{13}$$

Then

$$4X+A = (2Y+B)(2Q) + (4R+A-2BQ). \tag{14}$$

By examining the final term in parentheses in equation (14) 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 (12) and (13) we find

$$\frac{X}{Y} - \frac{Y-1}{Y} \leq Q \leq \frac{X}{Y} , \tag{15}$$

and from equations (9) and (10),

$$\frac{4X+A}{2Y+B} - \frac{2Y+B-1}{2Y+B} \leq QUOT \leq \frac{4X+A}{2Y+B} . \tag{16}$$

Combining these, we have after some algebra that

$$-2 + \frac{2BX + 2B + Y(4-A)}{Y(2Y+B)} \leq 2Q - QUOT \leq 1 + \frac{2XB - Y(A+1)}{Y(2Y+B)} . \tag{17}$$

When $B = 0$, the fact that the bounded quantity is an integer leads directly to the inequality

$$-1 \leq 2Q - QUOT \leq 0 \quad (B = 0). \tag{18}$$

When $B = 1$, we can use (11) to obtain a bound for X which allows the right side of (17) to be bounded above by

$$1 + \frac{M}{2Y} - \frac{A+1}{2Y} ,$$

and since $Y \geq M/4$ and $(A+1)/2Y > 0$, we have finally

$$-1 \leq 2Q - QUOT \leq 2 \quad (B = 1). \tag{19}$$

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

DIVIDEND	DIVISOR	B	2Q-QUOT
$\frac{1}{2}M^2 - M$	$\frac{1}{2}M+1$	1	2
$M-1$	$\frac{1}{2}M+1$	1	-1
$\frac{1}{2}M^2 - M$	$\frac{1}{2}M$	0	0
$M-1$	$\frac{1}{2}M$	0	-1

A flow diagram for the overall division process is given in Figure 2.

IV. Tests of the Algorithms

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 (9), 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 (18) and (19) were verified, and the algorithm is known to be correct.

V. A System/360 Implementation of the Division Algorithm

To illustrate the coding of the division scheme on an actual machine, a code sequence is given in Figure 3 which will perform the division algorithm on an IBM System/360 computer. 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. Acknowledgments

The author wishes to thank M. D. McIlroy and the referee for several helpful suggestions concerning the presentation of the material.

VII. References

1. Stanford Linear Accelerator Center Computation Group Technical Memorandum No. 18, "A Multiple Precision Floating Point Subroutine Package for System/360".

2. IBM System/360 Principles of Operation, File No. S360-01, Form A22-6821, International Business Machines Corporation.

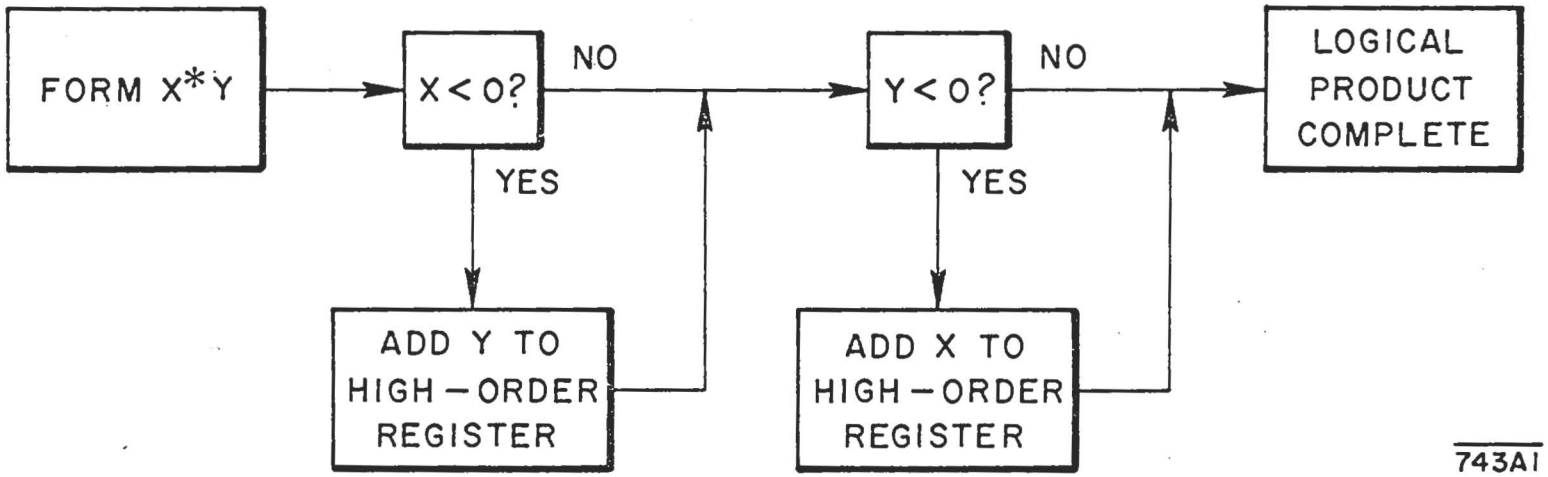

FIG. 1 -- LOGICAL MULTIPLICATION

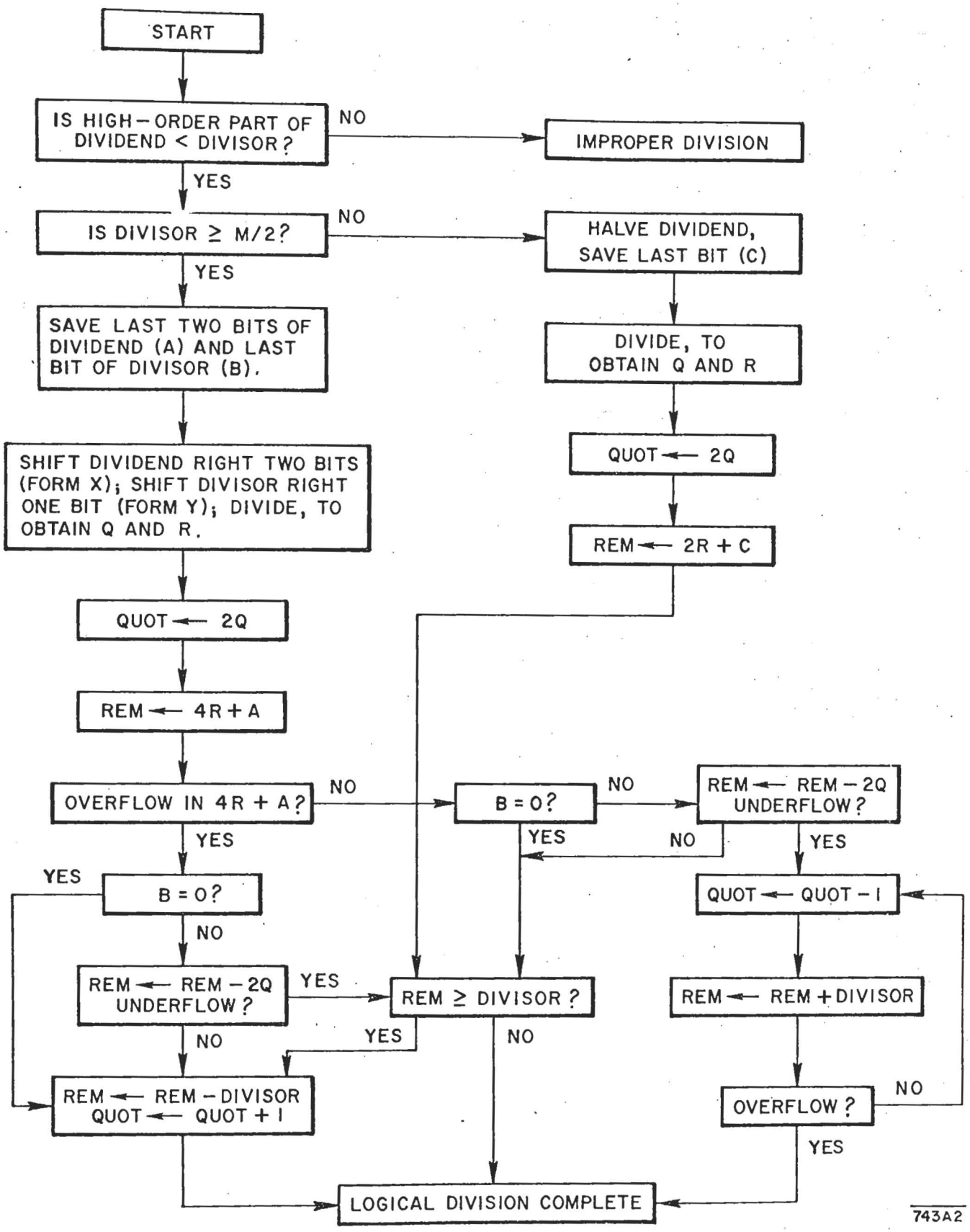

FIG. 2 -- LOGICAL DIVISION

```
        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