APPLICATIONS OF DIVISION BY CONVERGENCE

by

ROBERT E. GOLDSCHMIDT

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1964

Signature of Author .....
Department of Electrical Engineering, May 1964

Certified by .....
Thesis Supervisor

Accepted by .....
Chairman, Departmental Committee on Graduate Students

APPLICATIONS OF DIVISION BY CONVERGENCE

by

ROBERT ELLIOTT GOLDSCHMIDT

Submitted to the Department of Electrical Engineering on May 22, 1964 in partial fulfillment of the requirements for the degree of Master of Science.

ABSTRACT

A unique method of accomplishing the floating point divide instruction is investigated. The divisor and dividend are considered to be the denominator and numerator of a fraction. On each iteration, a factor multiplies both numerator and denominator such that the resultant denominator converges quadratically towards one (1) and the resultant numerator converges quadratically towards the desired quotient.

The paper develops the theory behind the process, applies it to the high performance computer area, and then evaluates the round off error and compares it with that of standard division methods. Finally, the criteria for applicability to other areas are determined.

The high performance application is the most significant development of the paper. By means of the convergence method, the divide execution time can be reduced to half that of conventional methods.

Thesis Supervisor: Alfred K. Susskind
Title: Associate Professor of Electrical Engineering

ACKNOWLEDGEMENT

The motivation for this investigation was provided by Dr. T. C. Chen of IBM who suggested that the division execution time could be reduced by using a type R_{BK} approximate reciprocal.

I wish to thank Professor Alfred K. Susskind who was my thesis supervisor and Mrs. M. Dearden who typed this paper.

THIS WORK WAS DONE UNDER THE AUSPICES OF THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY AND THE INTERNATIONAL BUSINESS MACHINES CORPORATION.

TABLE OF CONTENTS

SYMBOL DEFINITIONS

C_K	The length of that part of the intermediate denominator which extends to the right of the predicted minimum string and to the left of that portion which is not used to determine an approximate reciprocal.
D_K	The denominator which is the result of the K'th iteration.
E_{IK}	The round off error which is incurred in the formation of D_K from $D_{(K-1)}$.
E_{NIK}	The round off error which is incurred in the formation of N_K from $N_{(K-1)}$.
L_K	The predicted minimum string length of D_K.
$M_{(2K-1)}$	The low order multiple added to the partial product on multiply iteration K.
M_{2K}	The high order multiple added to the partial product on multiply iteration K.
MG1	The gate illustrated in figures one and two which selects $M_{(2K-1)}$.
MG2	The gate illustrated in figures one and two which selects M_{2K}.
N_K	The numerator which is the result of the K'th iteration.
P_K	The partial product which is the result of multiply iteration K.
PP1, PP2	The two registers illustrated in figures one and two which in combination store P_K.
PR1, PR2	The two registers illustrated in figures one and two which store the final P_K (the product).

$(1 + X_K)$	The 2^0 through $2^{-(L_K + C_K)}$ positions of D_K.
Y_K	The $2^{-(L_K + C_K + 1)}$ through 2^{-S} positions of D_K.
2^{-S}	The least significant position of the internal fractions.
R_{AK}, R_{BK}, R_{CK}	Three different types of approximate reciprocals which are determined from D_K.

APPLICATIONS OF DIVISION BY CONVERGENCE

PART ONE: THEORY

The author proposes the use of quadratically converging algorithms which utilize fast multiplication techniques in the execution of division. Although the amount of computation in this method is slightly greater than standard division practices, there is a speed advantage for several reasons.

Successive additions can be accomplished much more rapidly in "fast multiply" schemes than in a carry propagate adder. Secondly, in standard methods there is an interaction between the adder output and the control logic after each addition, while in the methods proposed, this interaction takes place only after the completion of a multiplication. Finally, since the convergence is quadratic, the number of iterations is proportional to $\log_2$ of the fraction length. This further reduces the number of data-control interactions.

Because this process does not yield a remainder, it will be applied only to floating point division. We will assume that the input fractions are positive and are of value less than one and greater than or equal to one half. We then consider:

$$\frac{\text{dividend}}{\text{divisor}} = \frac{\text{numerator}}{\text{denominator}} = \frac{N}{D} = \text{quotient}$$

Any factor which multiplies both numerator (N) and denominator (D) will leave the value of the quotient unchanged. In fact, if multiplication of both N and D by identical sequences of factors yields a denominator equal to one, then the resultant numerator will be identical to the quotient.

If we define the initial denominator (D_0) to be equal to $(1 + X_0 + Y_0)$, then $1 - (X_0 + Y_0) + (X_0 + Y_0)^2 - (X_0 + Y_0)^3 + \ldots$ is a single factor which if multiplied times N would yield the desired quotient. We may truncate this Taylor series at the point where $/(X_0 + Y_0)^K/$ is smaller than the least significant position of the characteristic.

Thus it can be shown that the convergence process takes a finite amount of time. The above series is not efficiently generated by hardware. A more efficient sequence of factors whose product is identical to the above series is:

$$\left[1 - (X_0 + Y_0)\right]\left[1 + (X_0 + Y_0)^2\right]\left[1 + (X_0 + Y_0)^4\right]\left[1 + (X_0 + Y_0)^8\right]\ldots$$

Another way of picturing this process is to consider the sequence of fractions shown below:

D_K is the K'th denominator

N_K is the K'th numerator

R_{AK} is the K'th factor in the above product

initial $$\frac{N_0}{D_0} = \frac{N_0}{1 + (X_0 + Y_0)} = \frac{N_0 \times R_{A0}}{[1 + (X_0 + Y_0)] \times R_{A0}} = \frac{N_0 \times [1 - (X_0 + Y_0)]}{[1 + (X_0 + Y_0)] \times [1 - (X_0 + Y_0)]} =$$

$$\frac{N_1}{1 - (X_0 + Y_0)^2} = \frac{N_1}{D_1}$$

general $$\frac{N_K}{D_K} = \frac{N_K}{1 - (-X_0 - Y_0)^{2^K}} = \frac{N_K \times R_{AK}}{[1 - (-X_0 - Y_0)^{2^K}] \times R_{AK}} =$$

$$\frac{N_K [1 + (-X_0 - Y_0)^{2^K}]}{[1 - (-X_0 - Y_0)^{2^K}] [1 + (-X_0 - Y_0)^{2^K}]} = \frac{N_{K+1}}{1 - (-X_0 - Y_0)^{2^{K+1}}} =$$

$$\frac{N_{K+1}}{D_{K+1}}$$

Note that it is possible to define $R_{AK} = (2 - D_K)$. This means that on a given iteration the computer takes inputs D_K, N_K derives R_{AK} and then forms $N_K \times R_{AK} = N_{K+1}$ and $D_K \times R_{AK} = D_{K+1}$. The process need only be continued until $/(X_0 + Y_0)^{2^K}/$ is smaller than the least significant position of the fractions.

For a 56 bit fraction this process would require six iterations or twelve multiplies. By using sufficient hardware it would be possible to perform the two multiplies for each iteration in parallel. Even so, the divide would take on the order of six multiply times while standard methods operate in as fast as four

multiply times. The process defined above does not take advantage of the fact that it is possible to shorten the length of R_K. In order to show this we must explain the format in which the numbers will appear.

	2^0	2^{-1}	2^{-2}	2^{-3}	... 2^{-LK}	$2^{-(LK+1)}$	... 2^{-2LK}	$2^{-(2L_{K+1})}$	... 2^{-5}
D_0, N_0	0	1	X	X	... X	X	... X	X	... X
D_K is	0	1	1	1	... 1	X	... X	X	... X
or	1	0	0	0	... 0	X	... X	X	... X
					$(1+X_K)$			Y_K	

The 2^{-5} position will be chosen small enough that no bits are lost during the operation. Later we will show how the format length can be reduced.

If we predict that $(1 - 2^{-LK}) \leq D_K \leq (1 + 2^{-LK} - 2^{-5})$ we then know that the format of D_K is one of the two shown above. D_K will have a leading string of at least L_K zeroes or ones. Because of this we will call L_K the predicted minimum string length. The converse is also true. If D_K has L_K minimum string length, then it is in the above range.

We will define two operations on numbers of the above format. The bit by bit complement of the number equal to (2 - number - 2^{-5}) and the "twos" complement of the number which is the bit by bit complement plus 2^{-5}. Thus the "twos" complement equals (2 - number).

The author will now proceed to define three approximate reciprocals R_{AK}, R_{BK}, and R_{CK}. On a particular iteration we will determine the approximate reciprocal from D_K and then multiply it by both D_K and N_K to form D_{K+1} and N_{K+1}: For each approximate reciprocal, we will derive the rate of convergence in terms of the predicted minimum string length, L_K.

we define $D_K = 1 + X_K + Y_K$

then $(1 - 2^{-L_K}) \leq (1 + X_K + Y_K) \leq (1 + 2^{-L_K} - 2^{-S})$ (1)

and $-2^{-L_K} \leq (X_K + Y_K) \leq (2^{-L_K} - 2^{-S})$ (2)

Convergence of R_{AK}

R_{AK} is defined to be the two's complement of D_K.

Thus $R_{AK} = (2 - D_K) = 1 - (X_K + Y_K)$

Note that this is the same approximate reciprocal used in the original convergence process. We now proceed to derive the rate of convergence.

$$D_K \times R_{AK} = \left[1 + (X_K + Y_K)\right] \times \left[1 - (X_K + Y_K)\right] = D_K + 1$$

$$D_K + 1 = \left[1 - (X_K + Y_K)^2\right] = D_K + 1$$

but from (2) we have $0 \leq (X_K + Y_K)^2 \leq 2^{-2L_K}$

thus $(1 - 2^{-2L_K}) \leq 1 - (X_K + Y_K)^2 \leq 1$

therefore $(1 - 2^{-2L_K}) \leq D_K + 1 \leq 1$

and certainly $(1 - 2^{-2L_K}) \leq D_K + 1 \leq 1 + 2^{-2L_K} - 2^{-S}$

we conclude that $L_{K+1} = 2L_K$ for R_{AK}

Convergence of R_{BK}

Both R_{BK} and R_{CK} are determined from $(1 + X_K)$, a number defined in the format explanation which has its least significant position 2^{-2L_K}. Y_K, which is used in the following derivations has its only non-zero bits in positions $2^{-(2L_K+1)}$ through 2^{-S}.

It can be shown that

$$0 \leq Y_K \leq (2^{-2L_K} - 2^{-S}) \tag{3}$$

$$(1 - 2^{-L_K}) \leq (1 + X_K) \leq (1 + 2^{-L_K} - 2^{-2L_K}) \tag{4}$$

R_{BK} is defined to be the one's complement of $(1 + X_K)$.

Thus $R_{BK} = 2 - (1 + X_K) - 2^{-2L_K} = 1 - X_K - 2^{-2L_K}$

We now proceed to derive the rate of convergence:

$$\begin{aligned} D_{K+1} = D_K \times R_{BK} &= [1 + (X_K + Y_K)][(1 - X_K - 2^{-2L_K})] \\ &= 1 + (Y_K - 2^{-2L_K}) - (X_K + Y_K)(X_K + 2^{-2L_K}) \end{aligned}$$

We will now determine the range of the second and third terms of this expression separately. If we subtract 2^{-2L_K} from all terms of range (3) we get

$$-2^{-2L_K} \leq (Y_K - 2^{-2L_K}) \leq -2^{-S} \tag{5}$$

We now analyze the third term. From range (2) we know that

$$-2^{-L_K} \leq (X_K + Y_K) \leq (2^{-L_K} - 2^{-S}) \tag{6}$$

From range (4) we know that

$$(1 - 2^{-L_K}) \leq (1 + X_K) \leq 1 + 2^{-L_K} - 2^{-2L_K}$$

$$-2^{-L_K} + 2^{-2L_K} \leq (X_K + 2^{-2L_K}) \leq 2^{-L_K} \tag{7}$$

Both the term $(X_K + Y_K)$ and $(X_K + 2^{-2L_K})$ contain X_K which has no non-zero bits after the 2^{-2L_K} position. In the first term we add something which is greater than zero and smaller than 2^{-2L_K}. (See (3)). In the second term we add 2^{-2L_K} into the least significant possible non-zero position of X_K. We can therefore conclude that when $(X_K + Y_K) < 0$, $(X_K + 2^{-2L_K}) \leq 0$

and when $(X_K + Y_K) \geq 0$, $(X_K + 2^{-2L_K}) > 0$

The product of the two terms is then always greater than zero.

From range (6) and (7) we can then show that $0 < (X_K + Y_K) * (X_K + 2^{-2L_K}) \leq (2^{-2L_K} - 2^{-3L_K})$ (8)

From this we can determine an upper and lower bound for D_{K+1}.

One	1	$\leq$	1	$\leq$ 1
plus (5)	$-2^{-2L_K} \leq (Y_K - 2^{-2L_K})$			$\leq -2^{-S}$
minus (8)	$(-2^{-2L_K} + 2^{-3L_K}) \leq -(X_K + Y_K)(X_K + 2^{-2L_K})$			< 0
$\therefore D_{K+1}$	$(1 - 2^{-(2L_K - 1)} + 2^{-3L_K}) \leq D_{K+1}$			$\leq (1 - 2^{-S})$

where $D_{K+1} = 1 + (Y_K - 2^{-2L_K}) - (X_K + Y_K)(X_K + 2^{-2L_K})$

then certainly $(1 - 2^{-(2L_K - 1)}) \leq D_{K+1} \leq 1 + 2^{-(2L_K - 1)} - 2^{-S}$

we conclude that $L_{K+1} = (2L_K - 1)$ for R_{BK}.

Convergence of R_{CK}

R_{CK} is defined to be the two's complement of $(1 + X_K)$.

Thus $R_{CK} = 2 - (1 + X_K)$

We now proceed to derive the rate of convergence:

$$D_{K+1} = D_K \times R_{CK} = (1 + X_K + Y_K)(1 - X_K)$$

$$= 1 + Y_K - X_K(X_K + Y_K)$$

From (3) $0 \leq Y_K \leq 2^{-2LK} - 2^{-S}$ (9)

From (4) $-2^{-LK} \leq X_K \leq (2^{-LK} - 2^{-2L_K})$

$$-2^{-LK} \leq (X_K + Y_K) \leq (2^{-LK} - 2^{-S})$$

By the format description one can see that the addition of Y_K does not affect the sign of X_K. Thus the terms X_K and $(X_K + Y_K)$ must have the same sign. We then can state that $0 \leq X_K(X_K + Y_K) \leq 2^{-2LK}$. By adding terms we can now determine an upper and lower bound for D_{K+1}.

One	$1 =$	≤ 1	≤ 1
plus (9)	0	$\leq Y_K$	$\leq 2^{-2LK} - 2^{-S}$
plus -(10)	-2^{-2LK}	$\leq -X_K(X_K+Y_K)$	≤ 0
D_{K+1}	$1-2^{-2LK}$	$\leq D_{K+1}$	$\leq 1 + 2^{-2LK} - 2^{-S}$

where $D_{K+1} = 1 + Y_K - X_K(X_K + Y_K)$

we conclude that $L_{K+1} = 2L_K$ for R_{CK}

Convergence of R_{CK} generalized

R_{CK} is the approximate reciprocal used in the divide execution. In some situations it becomes more efficient for the hardware to accomplish less than a doubling of the L_K and as a result further reduce the number of positions in R_{CK}. This is because the length of R_{CK} which can be multiplied times D_K or N_K in a time slot has an upper limit. For this purpose we will define a generalized R_{CK} and $(1 + X_K)$ as shown in the format description below:

	2^0	2^{-1}	2^{-2}	. . . .	2^{-L_K}	$2^{-(L_K+1)}$ . .	$2^{-(L_K+C_K)}$	$2^{-(L_K+C_K+1)}$ . . .	2^{-S}
D_K is	0	1	1	. . . .	1	X . . . X	X	. . .	X
or	1	0	0	. . . .	0	X . . . X	X	. . .	X
						$(1 + X_K)$		Y_K	

Where $C_K \leq L_K$

We wish to prove that $L_{(K+1)} = (L_K + C_K)$. From above format:

$$(1 - 2^{-L_K}) \leq (1 + X_K) \leq (1 + 2^{-L_K} - 2^{-(L_K+C_K)}) \qquad (11)$$

$$0 \leq Y_K \leq (2^{-(L_K+C_K)} - 2^{-S}) \qquad (12)$$

R_{CK} generalized $\triangleq$ two's complement of $(1+X_K)$ $= (1 - X_K)$

$$\therefore \quad D_{K+1} = D_K \times R_{CK} = (1 + X_K \mp Y_K)(1 - X_K)$$

$$\underline{D_{K+1} = 1 + Y_K - X_K (X_K + Y_K)}$$

The range of the term Y_K is given in (12).

From identical arguments as in the R_{CK} derivation, X_K has the same sign as $(X_K + Y_K)$.

from (11) $\quad -2^{-L_K} \leq X_K \leq 2^{-L_K} - 2^{-(L_K + C_K)}$

from (2) $\quad -2^{-L_K} \leq (X_K + Y_K) \leq 2^{-L_K} - 2^{-S}$

Thus $\quad 0 \leq X_K (X_K + Y_K) \leq 2^{-2L_K} \qquad (13)$

By adding we can now determine an upper and lower bound for D_{K+1}

One	1	$\leq$	1	$\leq$ 1
plus (12)	0	$\leq$	Y_K	$\leq$ $2^{-(L_K+C_K)} = 2^{-S}$
plus (13)	-2^{-2L_K}	$\leq$	$-X_K(X_K+Y_K)$	$\leq$ 0
$\therefore D_{K+1}$	$1-2^{-2L_K}$	$\leq$	D_{K+1}	$\leq 1 + 2^{-(L_K+C_K)} - 2^{-S}$

(14)

where $D_{K+1} = 1 + Y_K - X_K (X_K + Y_K)$

and certainly $\quad 1 - 2^{-(L_K+C_K)} \leq D_{K+1} \leq 1 + 2^{-(L_K + C_K)} - 2^{-S}$

We conclude that $\quad L_{K+1} = (L_K + C_K)$ for R_{CK}

where $C_K \leq L_K$

Description of R_{S0}

On the first iteration it is possible to determine a special reciprocal R_{S0} from inspection of positions 2^{-2} through 2^{-6} of D_0 such that the product $(1 - 2^{-6}) \leq (D_0 \times R_{S0}) \leq (1 + 2^{-6} - 2^{-S})$. This significantly reduces the number of iterations needed for convergence. The table giving the R_{S0}, D_0 pairs is shown below.

RECIPROCAL DECODER

Denominator in 64th's	Reciprocal Range in 256th's
$32 \leq D_0 < 33$	504
$33 \leq D_0 < 34$	489
$34 \leq D_0 < 35$	475
$35 \leq D_0 < 36$	461-462
$36 \leq D_0 < 37$	448-449
$37 \leq D_0 < 38$	436-437
$38 \leq D_0 < 39$	425-426
$39 \leq D_0 < 40$	414-416
$40 \leq D_0 < 41$	404-405
$41 \leq D_0 < 42$	394-396
$42 \leq D_0 < 43$	384-386
$43 \leq D_0 < 44$	376-378
$44 \leq D_0 < 45$	367-369
$45 \leq D_0 < 46$	359-361
$46 \leq D_0 < 47$	351-354
$47 \leq D_0 < 48$	344-346
$48 \leq D_0 < 49$	336-339
$49 \leq D_0 < 50$	330-332
$50 \leq D_0 < 51$	323-326
$51 \leq D_0 < 52$	317-320

RECIPROCAL DECODER

Denominator in 64th's	Reciprocal Range in 256th's
$52 \leq D_0 < 53$	311-313
$53 \leq D_0 < 54$	305-308
$54 \leq D_0 < 55$	299-302
$55 \leq D_0 < 56$	294-297
$56 \leq D_0 < 57$	288-291
$57 \leq D_0 < 58$	283-286
$58 \leq D_0 < 59$	279-282
$59 \leq D_0 < 60$	274-277
$60 \leq D_0 < 61$	269-272
$61 \leq D_0 < 62$	265-268
$62 \leq D_0 < 63$	261-264
$63 \leq D_0 < 64$	256-260

PART 2

THE MULTIPLY-DIVIDE UNIT

The first application of this algorithm will be in the area of a high performance Multiply-Divide unit. Since each division iteration utilizes multiply it is important that we understand first how the Multiply operation would be accomplished.

In both multiply and divide we will have 56 bit input fractions which have value greater than or equal to one half and less than one. The output fraction will be 57 bits (a 2^{o} position is added) and will have a range of greater than or equal to one fourth and less than two. Because of the divide round off error, the format used internal to the unit will contain position 2^{o} through 2^{-61}.

Multiply

Figure I shows only those elements which are necessary for a high speed multiply unit. The four input-two output adder is used to accumulate the multiplicand multiples in a minimal amount of time. After the final product is available it is fed into the two input, carry propagate adder where the single output product is developed. A multiplier encoder examines the low order bits of the multiplier and determines the two multiples to be added in on each pass through

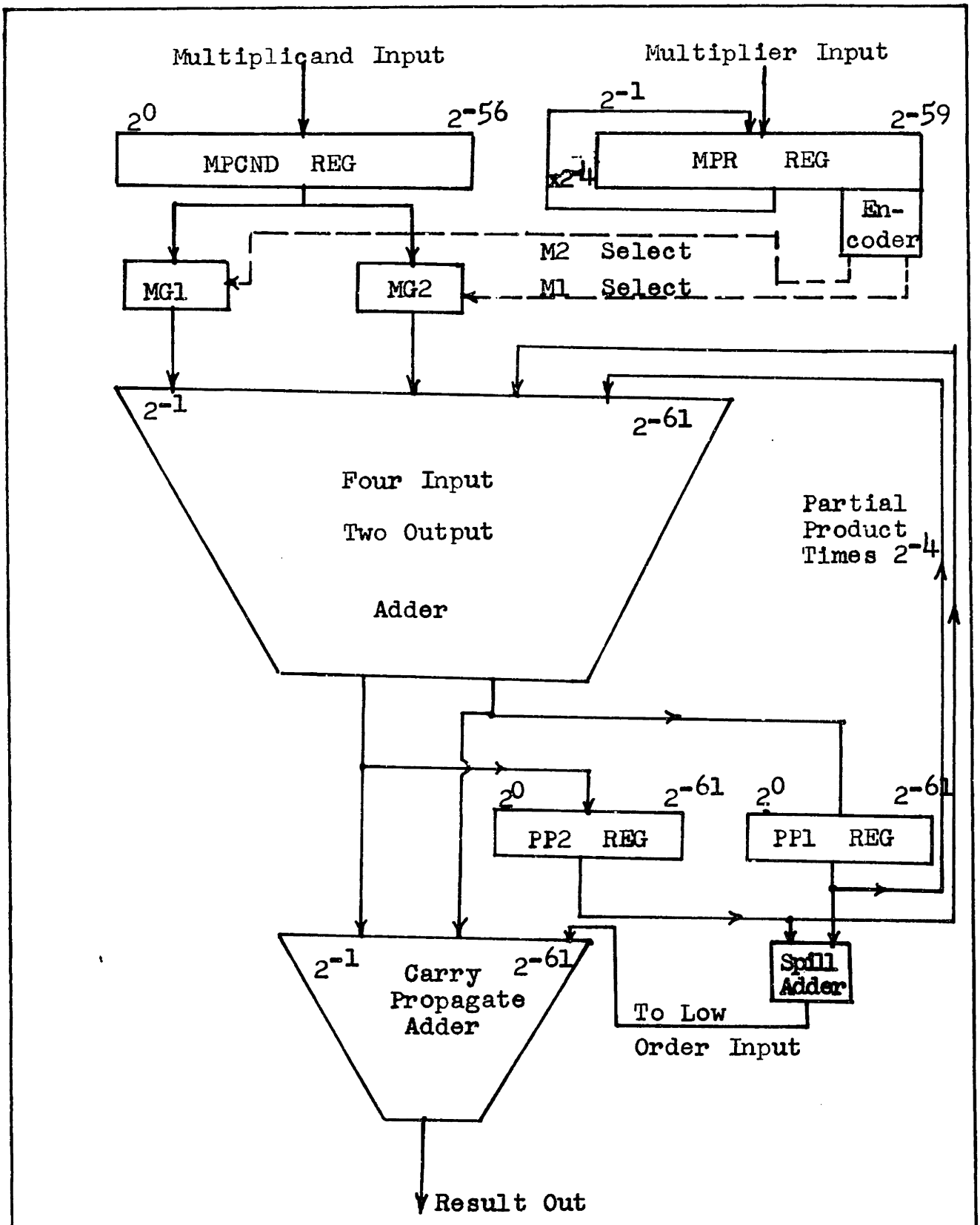

Figure One

The Multiply Unit

the four input adder. The multiplier (MPR) register is then shifted right four so that the next set of multiples can be determined. This process continues until the last set of multiples has been added in.

By allowing both positive and negative multiplicand multiples there is a method of encoding the multiplier so that a single power of two represents two bits of the multiplier. The encoding utilizes the identity $2^{-n}-2^{-(n+m)}=2^{-(n+1)}+2^{-(n+2)}\ldots 2^{-(n+m)}$. The encoding of the multiplier is accomplished by a set of three bit overlapping scans as shown below. This reduces the maximum possible number of non-zero multiple additions by a factor of two.

$$2^0{}_2{}^{-1}{}_2{}^{-2}{}_2{}^{-3}{}_2{}^{-4}\ldots 2^{-50}{}_2{}^{-51}{}_2{}^{-52}{}_2{}^{-53}{}_2{}^{-54}{}_2{}^{-55}{}_2{}^{-56}{}_2{}^{-57}{}_2{}^{-58}{}_2{}^{-59}$$

MPR=0 1 X X X ...X X X X X X X O O O

M30 M29 M28...M5 M4 M3 M2 M1

$$MPR=M_{30}x2^{0}+M_{29}x2^{-2}+M_{28}x2^{-4}\ldots+M_{3}x2^{-54}+M_{2}x2^{-56}+M_{1}x2^{-58}$$

The value of each multiple relative to the high order position of the three position scan is given below in Table two.

Multiple Encode

2^0	2^{-1}	2^{-2}	Multiple
0	0	0	...0
0	0	1	$...+2^{-1}$
0	1	0	$...+2^{-1}$
0	1	1	$...+2^{0}$
1	0	0	$...-2^{0}$
1	0	1	$...-2^{-1}$
1	1	0	$...-2^{-1}$
1	1	1	...0

Table Two

Using this technique one of five multiples (0, $\pm 2^0$, $\pm 2^{-1}$) is encoded for every group of two bits.

On a typical iteration, the K'th partial product (P_K) is shifted right four and multiplicand multiples M_{2K} and $2^{-2}M_{(2K-1)}$ are added to form P_{K+1}. Thus $P_{K+1} = 2^{-4} P_K + \text{MPCND } (M_{2K} + 2^{-2} M_{(2K-1)})$ (14)

And

$$P_0 = 0$$

$$P_1 = \text{MPCND } (M_2 + 2^{-2}M_1)$$

$$P_2 = \text{MPCND } (M_4 + 2^{-2}M_3 + 2^{-4} M_2 + 2^{-6}M_1)$$

. = .

. = .

. = .

$$P_{15} = \text{MPCND } (M_{30}+2^{-2}M_{29} + 2^{-4}M_{28}+\ldots 2^{-56}M_2+2^{-58}M_1)$$

Since only 2^{-1} through 2^{-56} position of the product are needed, the bits which are shifted right past the 2^{-61} position are discarded. There is a circuit (the Spill Adder) which inspects the portions of the two registers (PP1 and PP2) shifted off to determine whether there is a ripple carry into the 2^{-61} position of the partial product.

Referring to Figure I again, we see that we can set Po to zero by resetting the PP1 and PP2 registers. After placing the multiplicand and multiplier in their respective registers, the multiply operation begins with the encoding of multiples one and two from positions 2^{-54} through 2^{-59} of the multiplier. These values are then used to choose

the proper multiples from multiple gate one (MG1) and multiple gate two (MG2). The multiples from MG1 have a wired shift of right two to give the $2^{-2} \times M_{(2K-1)} \times$ MPCND term of equation (14) while multiple gate two (MG2) output is not shifted to give the MPCND x M_{2K} term. Each multiple gate must be capable of gating shifted right one true, shifted right one complemented, shifted zero true, and shifted zero complemented. Arithmetic done in the adder section is of the two's complement form. Therefore negative multiples which are bit by bit complements must be accompanied by an additional low order one in the 2^{-61} position (the least significant adder input). Prior to every addition, the partial product represented by PP1 and PP2 is shifted right four (see equation (14)) and fed to two of the four inputs of the adder MPR.

After fifteen passes through the loop, the product is reduced from PP1 and PP2 to the result in the carry propagate adder. The multiply execution time is determined by the repetition rate of the four input adder loop. In the present design, the four input add can be performed three times per machine cycle. Thus six multiples can be added to a partial product in one cycle. Since there can be a maximum of 29 non-zero multiples in a 56 bit multiplication, five cycles are sufficient for the iteration portion. An additional cycle is necessary for initialization and the carry propagate addition. The total execution time within the multiply unit is 6 cycles.

Division

Divide was attached to the multiply unit with the object of using as much of the existing hardware as possible. At first it would appear that parallel operations on the numerator (N) and the denominator (D) would require duplicate hardware. However, by adding product registers PR1 and PR2 it became possible to have a multiply (the 4 input adder loop) and an add (the 2 input adder) occurring simultaneously.

Figure 2 is the block diagram of the multiply-divide unit. The incoming denominator (D_0) is placed in the multiplicand register where one cycle is taken to determine the value of R_{S0} from positions 2^{-2} through 2^{-6} of D_0. On the next cycle the multiply of (D_0) x (R_{S0}) is accomplished and the result is gated to the PR1 and PR2 register. From this point operations on N and D are executed concurrently. On the third cycle PR1 and PR2 are added to form D_1 (from which R_{C1} is determined). On the same cycle the numerator (N_0) is multiplied by R_{S0}. By using the method described below any reciprocal multiplication where $C_K \leq 11$ can be performed in one cycle.

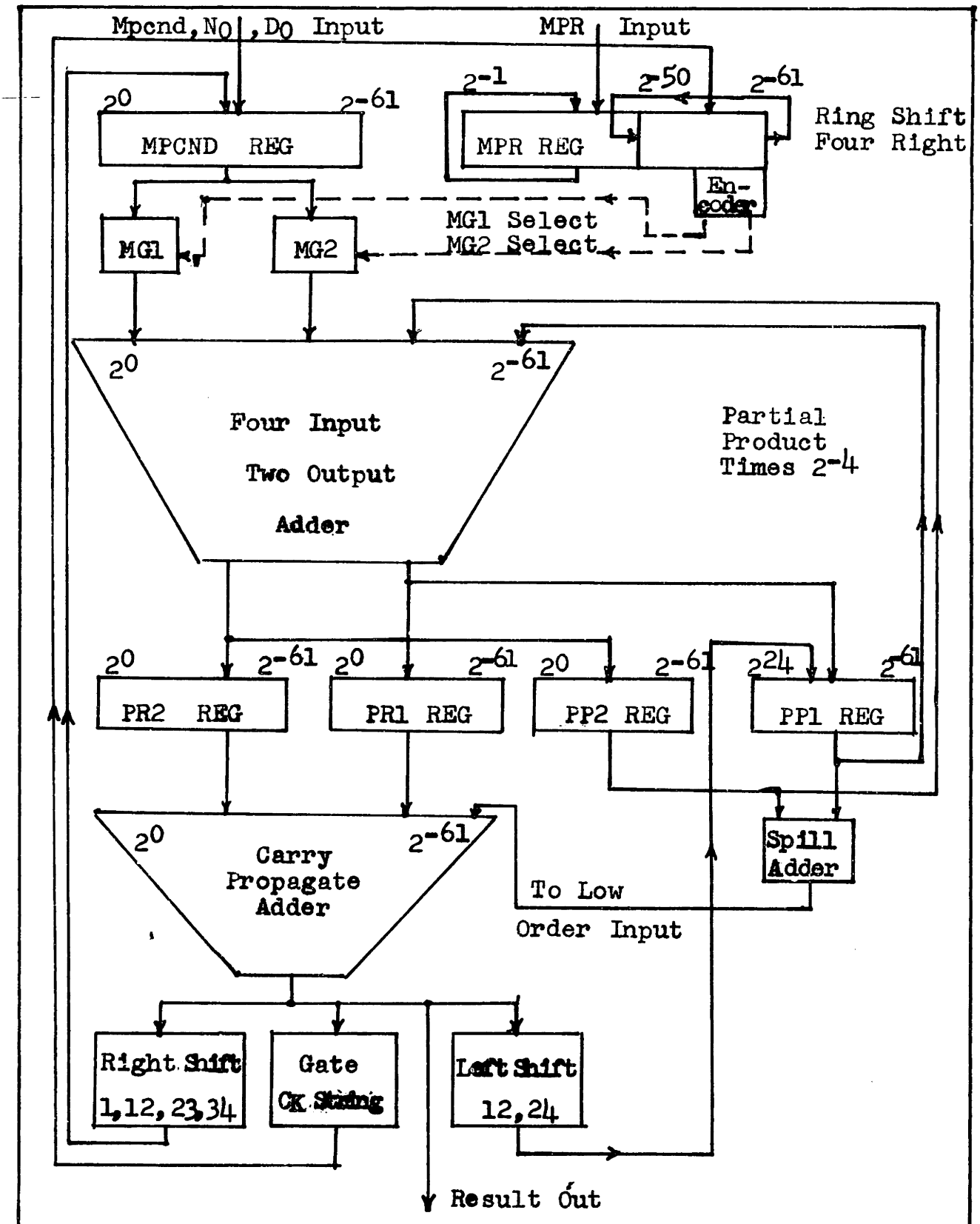

Figure Two

The Multiply--Divide Unit

One Divide Iteration

In order to understand how the R_{CK} multiplication is performed in one cycle we shall go back to the format definition of D_K.

D_K

$2^0\ 2^{-1} \ldots . 2^{-L_K}\ 2^{-(L_K+1)} \ldots . 2^{-(L_K+C_K)}\ 2^{-(L_K+C_K+1)} \ldots 2^{-61}$

is $1\ 0 \ldots . 0 \quad X \quad \ldots . X \quad X \quad \ldots X$

or $\underbrace{0\ 1 \ldots . 1 \quad X \quad \ldots . X}_{(1 + X_K)} \quad \underbrace{X \quad \ldots X}_{Y_K}$

We now recall that R_{BK} is the bit by bit complement of $(1+X_K)$. By forcing a one in the $2^{-(L_K+C_K+1)}$ position of R_{BK}, the encoded multiples will represent $R_{CK} = (R_{BK}+2^{-(L_K+C_K)})$. This can be deduced from inspection of Table 2. A maximum of six multiples can be added to the partial product in a cycle. Their alignment is shown below where L_K has the maximum single cycle value of 11.

$2^0\ 2^{-1} \ldots \quad 2^{-L_K} \quad\quad 2^{-(L_K+C_K)}$

R_{CK} is $0\ 1 \ldots 1\ 1 \quad XXXXXXXXXXX\ 1\ 0 \ldots 0$

or $1\ 0 \ldots 0\ 0 \quad XXXXXXXXXXX\ 1\ 0 \ldots 0$

$2^0 \quad M_6\ M_5\ M_4\ M_3\ M_2\ M_1$

All other multiples which could be encoded would be equal to zero. Thus:

$$R_{CK} = 1-X_K = 2^0+2^{-L_K}(M_6+2^{-2}M_5+2^{-4}M_4+2^{-6}M_3+2^{-8}M_2+2^{-10}M_1)$$

$$-X_K = 2^{-L_K}(M_6+2^{-2}M_5+2^{-4}M_4+2^{-6}M_3+2^{-8}M_2+2^{-10}M_1)$$

We recall from the multiply description that:

$$P_{K+1} = 2^{-4}P_K + M_{2K} + 2^{-2}M_{(2K-1)}$$

In the single machine cycle of R_{CK} multiplication, P_0 must be set equal to $2^{12}D_K$. This takes into account the 2^0 part of $R_{CK}=(2^0-X_K)$. The multiples which are added must contribute $D_K 2^{-L_K}(M_6+2^{-2}M_5 + 2^{-4}M_4+2^{-6}M_3+2^{-8}M_2+2^{-10}M_1) = -X_K$ towards the product (P_3). This is accomplished by placing $2^{-L_K} D_K$ in the MPCND register, and positions 2^{-L_K} through $2^{-(L_K + 12)}$ of R_{CK} right justified in the MPR register. The multiply sequence is then as follows:

initialization $PP_1 = 2^{12}D_K$

$PP_2 = 0$

MPCND $= 2^{-L_K} D_K$

MPR $= -X_K$ (right justified)

then $P_0 = 2^{12} D_K$

$P_1 = 2^8 D_K + 2^{-L_K} D_K (M_2 + 2^{-2} M_1)$

$P_2 = 2^4 D_K + 2^{-L_K} D_K (M_4 + 2^{-2} M_3 + 2^{-4} M_2 + 2^{-6} M_1)$

$D_{K+1} = R_{CK} \times D_K = P_3 = D_K + 2^{-L_K} D_K (M_6 + 2^{-2} M_5 + 2^{-4} M_4 + 2^{-6} M_3 + 2^{-8} M_2 + 2^{-10} M_1)$

The operand alignment for each iteration is listed below. On all iterations except the last, a one cycle multiply is performed. Provision must be made on all iterations, except the last, to multiply both N_K and D_K by R_K. This implies that the multiplier bits are not thrown away (e. g. , are ring shifted) as the multiples are encoded.

Iteration One - Initialization

$R_{SO} = 2_0 \quad 2^{-1}$

1 X X X X X X X X O O O

M_6 M_5 M_4 M_3 M_2 M_1

$2^1 D_0 = \text{MPCND}$

$PP_1 = PP_2 = O$

Iteration Two - Initialization

$$2^0 \quad 2^{-1} \qquad\qquad 2^{-12}$$

$$R_{C1} = \begin{array}{c} 0 \quad 1\ 1\ 1\ 1\ 1\ 1\ X\ X\ X\ X\ X\ X\ 1 \\ \underbrace{1}_{2^0} \quad \underbrace{0\ 0\ 0\ 0\ 0\ 0}_{M_6}\ \underbrace{X\ X}_{M_5}\ \underbrace{X\ X}_{M_4}\ \underbrace{X\ X}_{M_3}\ \underbrace{X\ X}_{M_2}\ \underbrace{X\ 1}_{M_1} \end{array}$$

$2^{-1}D_1 = \text{MPCND}$

$2^{-12}D_1 = PP_1 \qquad O = PP_2 \qquad L_1 = 6$

Iteration Three - Initialization

$$2^0 \quad 2^{-1} \ldots 2^{-12} \qquad\qquad 2^{-23}$$

$$R_{C2} = 0 \quad 1 \quad \ldots 1\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ 1$$

$$\underbrace{1}_{2^0} \quad 0 \quad \ldots \underbrace{0}_{M_6}\ \underbrace{X\ X}_{M_5}\ \underbrace{X\ X}_{M_4}\ \underbrace{X\ X}_{M_3}\ \underbrace{X\ X}_{M_2}\ \underbrace{X\ X\ 1}_{M_1}$$

$2^{-12}D_2 = \text{MPCND}$

$2^{12}D_2 = PP_1 \qquad 0 = PP_2 \qquad L_2 = 12$

Iteration Four - Initialization

$$2^0 \quad 2^{-1} \ldots 2^{-23} \qquad\qquad 2^{-34}$$

$$R_{C3} = 0 \quad 1 \quad \ldots 1\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ 1$$

$$\underbrace{1}_{2^0} \quad 0 \quad \ldots \underbrace{0}_{M_6}\ \underbrace{X\ X}_{M_5}\ \underbrace{X\ X}_{M_4}\ \underbrace{X\ X}_{M_3}\ \underbrace{X\ X}_{M_2}\ \underbrace{X\ X\ 1}_{M_1}$$

$2^{-23}D_3 = \text{MPCND}$

$2^{12}D_3 = PP_1 \qquad O = PP_2 \qquad L_3 = 23$

Iteration Five - Initialization

$$2^{0} \quad 2^{-1} \quad \ldots \quad 2^{-34} \qquad\qquad 2^{-57}$$

$$R_{C4} = 0 \quad 1 \quad \ldots \quad 1\ X\ X \ \ldots\ X\ X\ 1$$

$$1 \quad 0 \quad \ldots \quad \underbrace{0\ X\ X \ \ldots}_{M_{i2}\ \cdots\ M_2}\ \underbrace{X\ X\ 1}_{M_1}$$

$$\uparrow$$

$$2^{0}$$

$$2^{-34}D_3 = \text{MPCND}$$

$$2^{24}D_3 = PP_1 \qquad O = PP_2 \qquad L_4 = 34$$

The Divide Operation is accomplished as follows:

Cycle	Denominator	Numerator	$L_{K+1} = L_K + C_K$	$C_K \leq L_K$
1	Determine R_{S0}		$L_0 = 1$	Not applicable
2	Mpy $(D_0)x(R_{S0})$			
3	Add to form D_1, determine R_{C1}	Mpy $(N_0)x(R_{S0})$	$L_1 = 6$	$C_1 = 6$
4	Mpy $(D_1)x(R_{C1})$	Add to form N_1		
5	Add to form D_2, determine R_{C2}	Mpy $(N_1)x(R_{C1})$	$L_2 = 12$	$C_2 = 11$
6	Mpy $(D_2)x(R_{C2})$	Add to form N_2		
7	Add to form P_3, determine R_{C3}	Mpy $(N_2)x(R_{C2})$	$L_3 = 23$	$C_3 = 11$
8	Mpy $(D_3)x(R_{C3})$	Add to form N_3		
9	Add to form D_2, determine R_{C4}	Mpy $(N_3)x(R_{C3})$	$L_4 = 34$	$C_4 = 23$

Cycle	Denominator	Numerator	$L_{K+1} = L_K + C_K$	$C_K \leq L_K$
10		Add to form N_4		
11		Mpy $(N_4)x(R_{C4})$		
12		Mpy $(N_4)x(R_{C4})$		
13		Add to form N_5	$L_5 = 57$	

Note that D_4 is the last denominator that must be found in order to determine the quotient N_5. The accuracy of the quotient will be determined in the next section. The divide operation using current technology would be on the order of 2 us. Technologies which will be available in the next few years should reduce this by at least a factor of two. The execution time is twice that of the multiply time where standard division methods are at best four multiply times.

PART 3

ROUND OFF ERROR

The Multiply-Divide unit described in the preceeding chapter does not contain all bits formed in the divide iterations. Analysis of the introduced error in division is the purpose of this chapter. For the result to have any meaning, the division process, as implemented, must have the ability to continue the convergence of successive D_K's toward one (1), even though round off error has been introduced. Once this is proven, an exact analysis of the error can be made.

In order to determine the round off error in one divide iteration, we must remember that there are no positions smaller than 2^{-61} in any of the registers or adders. This means that whenever D_K or N_K is shifted right into the MPCND register, every positive multiple gated to the adder may have a string of ones to the right of the 2^{-61} position which is never added to the partial product. This means that the partial product can be low by 2^{-61}. By not adding in a low order one with negative multiples, (this low order one is inhibited only on divide iterations where D_K or N_K is shifted right), the error range is kept negative. If the low order one was added, the error could be positive which would increase the final

round off error. Again this error has a maximum value of 2^{-61}. Since two multiples are added to the partial product on each pass, the maximum error is -2^{-60} on a pass. Thus

$$\text{Error of } P_0 = 0$$

$$-2^{-60} \leq \text{Error of } P_1 \leq 0$$

$$-2^{-64}-2^{-60} \leq \text{Error of } P_2 \leq 0$$

$$-2^{-68}-2^{-64}-2^{-60} \leq \text{Error of } P_3 \leq 0$$

In addition the partial product is not retained past the 2^{-61} position so that an additional string of ones to the right of this position is discarded. This string is shifted off as the partial product is fed back into the four input adder and can cause an additional error of -2^{-61}. No bits can be lost in the carry propagate addition. Thus an approximate range for the round off error in multiplying $D_K \times R_K$ to form D_{K+1} or $N_K \times R_K$ to form N_{K+1} is:

$$-3 \times 2^{-61} \leq E_I \leq 0$$

where E_I is the error of one iteration

Now that we have established the limit of the round off error resulting from the multiplication in one iteration, let us again consider the overall convergence of the process. Specifically, we would like to prove that the predicted minimum string length is not affected by E_I. This would mean that the process would continue the convergence of

$(D_K + E_I)$ towards one (1). Algebraicly we would like to prove

$$\text{that if } 1 - 2^{-L_K} \leq D_K \leq 1 + 2^{-L_K} - 2^{-61}$$

$$\text{then } 1 - 2^{-(L_K+C_K)} \leq (D_{K+1} + E_I) \leq 1+2^{-(L_K+C_K)} - 2^{-S} \qquad (16)$$

Since E_I is always negative, the lower limit in the above range is the only one in question. Remembering equation (14) we will choose the worst case where $C_K = L_K$. Then the lower limit occurs where $Y_K = 0$ and $X_K = -2^{-L_K}$.

$$2^0 \quad 2^{-1} \quad \ldots . 2^{-L_K} \; 2^{-(L_K+1)} \ldots \; 2^{-2L_K} \ldots \; 2^{-61}$$

Then D_K is 0 1 .... 1 0 ... 0 ... 0

R_{CK} is 1 0 .... $\underbrace{0 \quad 1}_{M_6}$ ... 1 1

All multiples encoded will be zero except for M_6 which will have value 2^o. Thus the last pass through the four input adder we will have:

	2^0	2^{-1}	2^{-2}	... 2^{-L_K}	$2^{-(L_K+1)}$	... 2^{-2L_K}	$-(2^{L_K+1})$	... 2^{-61}
$(2^{-4}P_2) = D_K =$	0	1	1	... 1	0	... 0	0	... 0
plus $2^0(2^{-L_K} D_K) =$	0	0	0	... 0	1	... 1	0	... 0
. equals D_{K+1}	0	1	1	... 1	1	... 1	0	... 0

As can be seen the E_I for this particular case is zero because no bits are lost. Thus $D_{K+1} = 1 - 2^{-2L_K}$ and $L_{K+1} = 2L_K$.

The proof of equation (16) means that the hardware as defined is capable of continuing the convergence of $(D_{K+1} + E_I)$ towards 1. (The predicted minimum string lengths are unaffected). It <u>does not</u> imply

that the product of $D_O R_{SO} R_{C1} R_{C2}'' R_{C3}''' R_{C4}''''$ (where the primes indicate that these quantities may be affected by round off error) is in the range of D5. There are three distinct errors affecting the final quotient:

1. The error caused by incomplete convergence.
2. The error in R_{C2}, R_{C3} and R_{C4} caused by round off errors encountered in multiplying the denominators.
3. The round off error accumulated in multiplying R_{SO}, R_{C1}, R_{C2}, R_{C3} and R_{C4} times the numerator.

In order to show the effect of round off error, a single prime will indicate the quantity may have been affected by E_{I1}, the round off error in the multiplication of $D_0 \times R_{S0}$. A double prime will indicate the quantity may have been affected by both E_{I1} and E_{I2} (the round off error in the multiplication of $D_I \times R_{CI}$). The triple and quadruple primes follow the same logic.

We begin the divide with D_0 from which we determine R_{S0}. When we multiply $D_0 \times R_{S0}$ the result is $D_I + E_I = D_1'$. From D_1' we determine R_{C1} which is not affected by E_{I1} because all bits of D_0

(a 56 bit number) were gated to the four input adder. Thus D_1' represents the 2^o through 2^{-61} positions of D_1.

$$-2^{-61} \leq E_{I1} \leq 0$$

Since R_{C1} is developed from positions 2^{-6} through 2^{-12} of D_1', its value is not affected by the round off error (see page). This is not true for the succeeding R_{CK}'s.

Thus $D_0 \times R_{S0} \longrightarrow D_1 + E_{I1} = D_1'$

$D_1' \times R_{C1} \longrightarrow D_2' + E_{I2} = D_2''$

$D_2'' \times R_{C2}'' \longrightarrow D_3'' + E_{I3} = D_3'''$

$D_3''' \times R_{C3}''' \longrightarrow D_4''' + E_{I4} = D_4''''$

$D_4'''' \times R_{C4}'''' \longrightarrow D_5''''$

There is no E_{I5} since this multiplication is never performed.

The final denominator

$$D^{5''''} = \left\{\left[\left((D_0 R_{S0} + E_{I1}) R_{C1} + E_{I2}\right) R_{C2}'' + E_{I3}\right] R_{C3}''' + E_{I4}\right\} R_{C4}''''$$

And since $L_4 = 34$ and $C_4 = 23$

We know from equation (14) that

$$1 - 2^{-68} \leq D_5'''' \leq 1 + 2^{-57} - 2^{-61}$$

Let us now determine how well R_{S0}, R_{C1}, R_{C2}'', R_{C3}'''' R_{C4}'''' approximates $\frac{1}{D_0}$. First let us assume that the only round off error is E_{I1}. We have proved that the process will continue to converge on each

iteration towards the reciprocal of the current denominator.

Thus $(D_0 R_{S0} + E_{I1}) R_{C1} R_{C2}' R_{C3}' R_{C4}' = D_5'$ (17)

and D_5' will lie in the same range as D_5''''

If we now assume that E_{I1} and E_{I2} are the only non-zero round off errors then $\left[(D_0 R_{S0} + E_{I1}) R_{C1} + E_{I2}\right] R_{C2}'' R_{C3}'' R_{C4}'' = D_5''$ (18)

It follows that from dividing equation (17) by (18)

that $R_{C2}' R_{C3}' R_{C4}' = R_{C2}'' R_{C3}'' R_{C4}'' \dfrac{(D_0 R_{S0} + E_{I1}) R_{C1} + E_{I2}}{(D_0 R_{S0} + E_{I1}) R_{C1}} \cdot \dfrac{D_5'}{D_5''}$ (19)

substituting in equation (17) we get

$$(D_0 R_{S0}+E_{I1}) R_{C1} R_{C2}'' R_{C3}'' R_{C4}'' \quad \frac{(D_0 R_{S0}+E_{I1}) R_{C1}+E_{I2}}{(D_0 R_{S0}+E_{I1}) R_{C1}} \cdot \frac{D_5'}{D_5''} = D_5'$$

recalling that $D_0 R_{S0} \simeq 1$

and $(D_0 R_{S0} + E_{I1}) R_{C1} \simeq 1$

and that $|E_{I2}| \leq 3 \times 2^{-61}$

and that $|E_{I1}| \leq 2^{-61}$

We approximate $\dfrac{(D_0 R_{S0}+E_{I1}) R_{C1}+E_{I2}}{(D_0 R_{S0}+E_{I1}) R_{C1}} \simeq \dfrac{1}{(1 - E_{I2})}$

and $(D_0 R_{S0} + E_{I1}) \simeq \dfrac{D_0 R_{S0}}{(1 - E_{I1})}$

And thus $D_0 R_{S0} R_{C1} R_{C2}'' R_{C3}'' R_{C4}'' = D_5'' (1 - E_{I1}) (1-E_{I2})$

Using similar reasoning it can be shown that:

$$D_0 R_{S0} R_{C1} R_{C2}'' R_{C3}''' R_{C4}'''' = D_5'''' (1-E_{I1})(1-E_{I2})(1-E_{I3})(1-E_{I4})$$

$$R_{S0} R_{C1} R_{C2}'' R_{C3}''' R_{C4}'''' = \frac{1}{D_0} \quad D_5''''(1-E_{I1})(1-E_{I2})(1-E_{I3})(1-E_{I4})$$

Now we will evaluate the third source of error; that encountered in multiplying the sequence of factors $R_{S0} R_{C1} R_{C2}'' R_{C3}''' R_{C4}''''$ times No. Here the round off errors on multiplications have the same range as E_{I1}, E_{I2}, E_{I3}, and E_{I4}.

Since the numerator round off errors are distince from the denominator we will denote them by E_{NI1}, E_{NI2}, E_{NI3}, and E_{NI4}. Since the errors in D_1, for example, are multiplied by R_{C1} which is approximately one, the numerator round off error is additive. Thus

$$N_5 = N_0 \left[\frac{1}{D_0} (D_5'''') (1-E_{I1}) (1-E_{I2}) (1-E_{I3}) (1-E_{I4}) \right]$$
$$+ E_{NI1} + E_{NI2} + E_{NI3} + E_{NI4} + E_{NI5}$$

recalling that $-2^{-61} \leq (E_{NI1} = E_{I1}) \leq 0$

and that $-3 \times 2^{-61} \leq (E_{NI2-5} = E_{I2-4}) \leq 0$

and that $1 - 2^{-68} \leq D_5'''' \leq 1 + 2^{-57} - 2^{-S}$

We conclude that

$$(1 - 10 \times 2^{-61}) \frac{N_0}{D_0} \leq N_5'''' \leq \left(\frac{N_0}{D_0} + 29 \times 2^{-61} \right)$$

or approximating $\left[(1 - 2^{-58}) \frac{N_0}{D_0} \right] \leq N_5 \leq \left[\frac{N_0}{D_0} + 2^{-56} \right]$

This is the range of accuracy for the 61 bit N_5''. Only positions 2^{-1} through 2^{-56} will be in the result and thus an additional 31×2^{-61} may be lost.

Thus $$\frac{N_0}{D_0}(1-2^{-58})-2^{-56} < \text{Result} < \frac{N_0}{D_0} + 2^{-56}$$

To get a comparison, standard division methods develop a result which is the high order 56 bits of the infinite length exact quotient. Thus the error range is:

$$\frac{N_0}{D_0} - 2^{-56} \leq \text{standard result} \leq \frac{N_0}{D_0}$$

The standard method error never alters any positions of the exact quotient which lie in the 2^{-1} through 2^{-56} positions, while the convergence method may. From the viewpoint of accuracy, however, the convergence method is in the same ball park.

PART FOUR

CRITERIA FOR APPLICABILITY

In order to make efficient use of the convergence method, the engineer must determine the pay off within his particular machine organization. The purpose of this section is to give a qualitative and quantitative evaluation of the relative merits of this method.

It must be remembered that there are several standard division methods whereby two bits of the quotient are generated each pass through a carry propagate adder. If we consider the time through the adder to be approximately a machine cycle, the divide operation takes half as many cycles as there are fraction positions.

If the arithmetic unit contains only a single carry propagate adder, the convergence method cannot be competitive. The total number of multiplier bits contained in all iterations of the convergence method can be shown to be approximately three halves of the input fraction length. By iterating at a rate of two multiplier bits per cycle (one multiple added to the partial product per cycle), the convergence method would take at least 50% longer.

Remembering that with the R_{CK} approximate reciprocal, C_K is the added string length on each operation, and also the length of the multiplier string. Then the sum of all C_K's is approximately equal to the input fraction length.

If the execution of the D_K and N_K multiplications can be accomplished concurrently, then the execution time is that of one full length multiplication plus one propagate add cycle for each iteration. If the execution of the D_K and N_K multiplications is not concurrent then the fastest execution time is realized when the predicted minimum string length is doubled on the last iteration. This means that three halves of a full multiply time is sufficient (assuming that no overhead cycles are associated with each multiply as in the scheme of figure one). Using this information we can now evaluate this method in several areas.

In the high performance area, we have already shown a factor of two over conventional methods. In a single adder arithmetic unit, the three-halves multiplication time is slower than conventional methods where the fastest divide can operate at two bits per cycle. This is because of the fact that the fastest multiply which has no overhead cycles is two bits per cycle.

The serial by character scientific computer area has a slow divide since restoring division methods are used (e. g. , the IBM 1620). Here an average of six subtractions are necessary to determine one decimal digit. The convergence method, being on the order of three halves of a multiply time, will give a speed advantage if the multiply time is less than two-thirds of the divide time.

It must be remembered, however, that the convergence method does not yield a remainder and fixed point divide must therefore be provided for with separate hardware or programming. Only a speed advantage may be gained by using the convergence method.

BIBLIOGRAPHY

1. C. S. Wallace, "A Suggestion for a Fast Multiplier," Trans IEEE, Vol. EC-13, Number 1, pp. 15, 16.

2. M. V. Wilkes, D. J. Wheeler, and S. Gill, "Preparation of Programs for an Electronic Digital Computer", Addison - Wesley, Cambridge, Mass.; 1951.

3. T. C. Chen, "Fast Division Scheme", private communication; November 4, 1963.

4. J. Cocke, G. M. Amdahl, F. B. Hartman, E. L. Willette, "Multiply-Divide Unit of Impact I ALU", IBM Disclosure 76,408, March 1963.