tive and can profit from the LSI advantages that iterativeness implies.

REFERENCES

[1] T. G. Hallin and M. J. Flynn, "Pipelining of arithmetic functions," *IEEE Trans. Comput.*, vol. C-21, pp. 880–886, Aug. 1972.
[2] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 14–17, Feb. 1964.
[3] S. F. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M. Powers, "Floating-point execution unit," *IBM J. Res. Develop.*, vol. 11, pp. 34–53, Jan. 1967.
[4] H. H. Guild, "Fully iterative fast array for binary multiplication and addition," *Electron. Lett.*, vol. 5, p. 263, June 12, 1969.
[5] J. Deverell, "Sequential generalized array," *Electron. Lett.*, vol. 8, pp. 9–10, Jan. 13, 1972.
[6] J. G. Earle, "Latched carry-save adder," *IBM Tech. Discl. Bull.*, vol. 7, pp. 909–910, Mar. 1965.
[7] T. G. Hallin, "Pipelining of arithmetic units," M.S. thesis, Dep. Elec. Eng., Northwestern University, Evanston, Ill., 1970.
[8] D. P. Burton and D. R. Noaks, "High speed iterative multiplier," *Electron. Lett.*, vol. 4, p. 262, June 28, 1968.
[9] J. Deverell, "The design of cellular arrays for arithmetic," *Radio Electron. Eng.*, vol. 44, pp. 21–26, Jan. 1974.

Parallel Multiplicative Algorithms for Some Elementary Functions

P. W. BAKER

***Abstract*—This correspondence presents generalized higher radix algorithms for some elementary functions which use fast parallel m-bit multipliers where radix $= 2^m$. These algorithms are extensions of those iterative schemes which are based on multiplications by $(1 + 2^{-i})$ and the use of prestored values of $\ln(1 + 2^{-i})$ and $\tan^{-1}(2^{-i})$. The particular functions under consideration are y/x, $y/x^{1/2}$, $y \cdot \exp(x)$, $y + \ln(x)$, $\sin(x)$ and $\cos(x)$ [and hence $\tan(x)$]. The extended algorithms rely on multiplication by $(1 + d_i r^{-k})$ where d_i, $0 \leq d_i < r$, is an m-bit integer. Using a simple selection procedure for d_i, simulations show that p(radix r) digits of a function may be generated, on the average, in less than $p + 1$ iterations.**

***Index Terms*—Continued products and sums, digital arithmetic, elementary functions, iterative algorithms, parallel m-bit multipliers.**

I. INTRODUCTION

A considerable amount of research has been directed towards refining a class of algorithms, based on continued product and sums, for the generation of the elementary functions y/x, $y/x^{1/2}$, $y \cdot \exp(x)$, $y + \ln(x)$, $\sin(x)$, and $\cos(x)$. Several publications [1]–[6] have described algorithms for binary arithmetic. These schemes work in radix 2 and require the operations of shifting, adding, and/or subtracting and the recall of prestored constants in order to generate 1 bit of the required function per iteration. De Lugish [5] has defined efficient algorithms which are based on a redundancy recoding technique used in fast division schemes. This technique requires a systematic 1-bit left shift of a partially converged result together with two 4-bit comparisons to select a ternary digit for the next iteration. This selection of digits reduces the average number of shifts and full precision additions to about $\frac{1}{3}$ of those required in the conventional schemes [4].

Manuscript received March 27, 1973; revised October 7, 1974. This work was supported by the Australian Research Grants Committee.
The author is with the Department of Computer Science, School of Electrical Engineering, University of New South Wales, Kensington, New South Wales, Australia.

Chen [6] uses a technique similar to skipping over zeros to generate some of the above mentioned algorithms in about one conventional multiply time. One conventional multiply time is that taken to multiply two n-bit numbers using an n-bit ripple adder with n single bit multiplications and parallel additions. Recently, Ercegovac [7] discussed a radix 16 digit by digit evaluation of quotients, logarithms, and exponentials. Digits are selected from the symmetric set $\{-10,-9,\cdots,9,10\}$ by a modified rounding procedure which requires the inspection of 7 bits of a partially converged result. For a given word length, Ercegovac has concluded [7] that these radix 16 algorithms will take about $\frac{3}{4}$ of the time required for the corresponding De Lugish algorithms.

The radix 2 digit by digit methods are considerably faster than polynomial approximation methods, when the multiplications required for the polynomial evaluations are executed in the conventional manner described above (see [3] for a detailed discussion). However, with the reduction in cost of medium scale integration (MSI) chips, higher radix multipliers, which retire several bits of multiplier per iteration, are economically feasible, even for small scale computers. Such multipliers can be several times faster than conventional multipliers, thereby allowing polynomial methods of function evaluation to compete with radix 2 digit by digit methods. Following the arguments in [3], the use of higher radix multipliers to generate several bits per iteration in digit by digit methods should allow these methods to stay ahead of polynomial evaluation. This correspondence presents generalized higher radix algorithms for the above mentioned functions which use a parallel m-bit multiplier where radix $= 2^m$. These higher radix algorithms generate m bits of the result per iteration and offer almost the same increase in speed over the conventional radix 2 algorithms as the higher radix multipliers offer over the conventional radix 2 multipliers.

II. HIGHER RADIX MULTIPLIERS

This section will confine itself to examining multipliers for radices which are an integral power of 2, namely,

$$r = 2^m \qquad m = 1,2,3,\cdots.$$

An n-bit number where

$$n = p \times m \qquad p \text{ integer}$$

can be regarded as a p digit number in radix $r = 2^m$ where each digit consists of m bits. An m-bit multiplier (BM), denoted by m BM is defined as a combinational logic circuit which multiplies an n-bit number by an m-bit number, $n > m$, to produce an $n + m$ bit product. For $n = 17$, the 40-ns Pezaris array multiplier [8] is a 17 BM. A practical realization of a general m BM could be formed from a subset of the carry-save multiplier scheme proposed in [9]. The m BM would use m rows of n full adders connected in a cascaded carry-save scheme, terminated by a carry propagate adder. In the simplest case a ripple adder would be used; but for faster execution, carry-lookahead adders may be constructed using MSI 4-bit adders and carry-lookahead generators. These MSI chips are currently available from several vendors.

A more compact scheme would make use of recently available 2×4 bit two's complement multipliers contained in one MSI package [10]. The logic on the chip incorporates a multiplier recoder and hence the number of rows of these devices required to form an m BM is equal to the smallest integer greater than $m/2$. For example, a 7 BM which would execute a $7 \times n$ bit multiplication would require 4 rows of packages. This 7 BM is illustrated in Fig. 1. In the limiting case of very large n, $n \gg m$ ($n \to \infty$, $m = 2$ to 8), the delay through this m BM will only be marginally slower than a 1 BM. This can be seen by examining the carry propagation path in Fig. 1. Since $n \gg m$, only a small portion of the total propagation time will be taken for the vertical delays. It follows then that multiplication

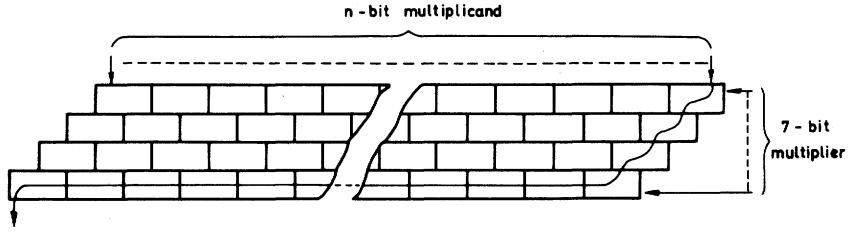

Fig. 1. A 7 BM which consists of 4 rows of 2-bit by 4-bit multiplier chips.

of two n-bit numbers using an m BM will be almost m times faster than multiplication using a 1 BM.

III. THE ALGORITHMS

To calculate $Q = Y/X$ ($\frac{1}{2} \leq X < 1$), where X, Y, and Q are n-bit words, Chen [6] multiplies X and Y by factors $(1 + 2^{-i})$, $1 \leq i \leq n$, such that X approaches unity and Y approaches Q with an error $\simeq 2^{-n}$. The factors are so chosen to reduce a multiplication to a shift and add sequence, the evaluation of Y/X requiring about n such sequences. For the evaluation of $Y + \ln X$ and Ye^X, Chen [6] uses a modification of Specker's method [4] which relies on appropriate multiplications by $(1 + 2^{-i})$ and the use of values of $\ln (1 + 2^{-i})$. $Y/X^{1/2}$ is calculated by multiplying X by $(1 + 2^{-i})^2$ and Y by $(1 + 2^{-i})$. For evaluating $\sin X$ and $\cos X$, Specker uses complex multiplications by $(1 + j2^{-i})$ and prestored values of $\tan^{-1}(2^{-i})$.

The following is an extension of these methods to radix $r = 2^m$. In what follows it will be assumed, unless otherwise stated, that Y and X are normalized n-bit fractions which consist of p digits radix 2^m, where $p = n/m$. The n bits do not include any guard digits added to maintain accuracy.

It is convenient to introduce the following notation.

A "mature" digit in radix $r = 2^m$ is defined as one whose value is $r - 1$. In binary, this means that the digit consists of m "ones," i.e.,

$$\underbrace{111\cdot\cdot11}_{m \text{ "ones"}} \triangleq r - 1.$$

An "equivocal" digit is one whose value is unknown.

Since the evaluation of Y/X is representative of the class of algorithms, it will be treated here in some detail.

A. Algorithm for Y/X

The following transformation is used:

$$Q = Y/X = \frac{Y \cdot \Pi f_i}{X \cdot \Pi f_i}, \qquad i \geq 1. \tag{1}$$

The f_i are selected to be of the form $(1 + d_i r^{-k})$, $d_i \in \{0,1,2,\cdots, r-1\}$ so that multiplication by f_i can be reduced to a shift and m-bit multiplication sequence. If

$$X \cdot \Pi f_i \to 1$$

then

$$Y \cdot \Pi f_i \to Q.$$

The multiplicative normalization of X, and the associated generation of Q, are performed recursively as follows.

Algorithm DIV:

$$X_1 = X, \qquad Y_1 = Y, \qquad i = 1, \qquad k = 1$$

$$X_{i+1} = X_i(1 + d_i r^{-k})$$

$$= X_i + d_i X_i r^{-k}, \qquad k \geq 1 \tag{2}$$

$$Y_{i+1} = Y_i(1 + d_i r^{-k}). \tag{3}$$

Assume that X_i has $k - 1$ leading mature digits, i.e.,

$$X_i = \overbrace{0.111\cdot\cdot11 \quad 111\cdot\cdot11 \quad \cdots \quad 111\cdot\cdot11}^{k-1 \text{ mature digits}} \quad xxx\cdot\cdot xx \quad \cdots$$

$$= 1 - r^{-(k-1)} + a_k r^{-k} + a_{k+1} r^{-(k+1)} + \cdots \tag{4}$$

where x denotes an equivocal bit and a_k, a_{k+1}, etc., are equivocal m-bit digits. It is well known [11] that the number of leading ones in X_i gives a measure of significance for Y_i. Hence Y_i will have about $(k - 1) \cdot m$ correct leading bits. d_i, $i > 1$ is now selected as

$$d_i = r - a_k - 1$$

$$= \text{one's complement of } a_k.$$

With this selection of d_i, the kth digit of X_{i+1}, after performing (2), may be mature or it may equal $r - 2$, this latter condition being called undershoot. The reader may verify the possibility of undershoot by performing (2) for the worst case, $a_k = a_{k+1} = \cdots = 0$ and, say, $r = 16$. This verification is more easily facilitated by writing $X_i r^{-k}$ as

$$X_i r^{-k} = \underbrace{0.000\cdots00}_{m\cdot k - 1 \text{ zeros}} \quad 1 \quad \underbrace{000\cdot\cdot00}_{m\cdot(k-1)-1 \text{ zeros}} \quad \bar{1} \quad 00\cdots$$

where $\bar{1} = -1$. Clearly, a necessary condition for undershoot is that $a_{k+1} = a_{k+2} = \cdots = a_{2k-2} = 0$. Hence the probability of undershoot decreases with increasing k. If no undershoot occurs, k is incremented by one. If undershoot occurs, the recursion on k is repeated to mature the kth digit of X_{i+2}. As with similar schemes, [11], [12], the first multiplier d_1 must be chosen by table lookup, i.e., $d_1 = f(X)$. When $k = p + 1$,

$$Q \simeq Y_{\text{last}}. \tag{5}$$

At the completion of Algorithm DIV, the error bound of the normalized X_{last} is

$$|\epsilon^D| \leq r^{-p}.$$

It follows that the maximum absolute error in Y_{last} is

$$|\epsilon^N| < 2 \cdot |\epsilon^D| = 2^{-n+1}.$$

The lookup table for d_1 will have as its input at least the m most significant bits of X after its leading 1 ($X \geq (0.1000\cdot\cdot)_2$). Assume that the input to the table is, say, the 6 most significant bits after the leading one. The entry d_1 for a given input, say $x_2x_3x_4x_5x_6x_7$, may be determined as follows.

$$\text{Form} \quad X_1 = 0.1x_2x_3x_4x_5x_6x_7 1111\cdots.$$

The correct entry will be the largest integer value of b for which

$$X_2 = X_1(1 + b \cdot 64^{-1})$$

is less than 1.

Algorithm DIV was simulated, with $Y = 1$, for radices 16, 32, 64, and 128 in order to estimate the number of undershoots that might occur after a large number of trials. 4000 trials were made for two separate cases. In the first, the values of X were varied

TABLE I

Radix	Method of Selecting X	Number of Undershoots 1	2	3	Total
$r = 16$	incremental	1022	3	0	1028
	random	754	1	0	756
$r = 32$	incremental	849	7	0	853
	random	861	0	0	861
$r = 64$	incremental	1005	1	0	1007
	random	1001	1	0	1003
$r = 128$	incremental	1362	3	0	1368
	random	1337	1	0	1339

incrementally from 0.5 to almost 1 by adding 0.000124937 to the last value of X. In the second case, the values of X were chosen from the interval $[\frac{1}{2},1)$ using a random number generator. The results of these simulations are given in Table I. In all cases, the input to the lookup table consisted of the 8 bits after the leading 1 of X.

For $r = 128$ and uniform incremental selection of X, Table I shows that out of 4000 examples, 1362 of them had one undershoot, 3 examples had 2 undershoots and no examples had 3 or more undershoots. According to Table I then, the generation of p digits of Q will take, on the average, less than $p + 0.4$ iterations. For $r = 128$, the number of undershoots would be reduced if a larger lookup table for d_1 were used.

B. Algorithm for $Y + \ln X$

Following Specker [4], we use the transformation

$$\ln X = \ln (X\Pi f_i) - \Sigma \ln f_i.$$

If

$$X\Pi f_i \to 1$$

then

$$Y - \Sigma \ln f_i \to Y + \ln X.$$

Argument range; $0.5 \leq X < 1$, $0.5 \leq Y < 1$. $Y + \ln X$ may be generated by Algorithm DIV with (3) replaced by (6);

$$Y_{i+1} = Y_i + L_k(d_i) \tag{6}$$

where $L_k(d_i)$ = two's complement of $\ln (1 + d_i r^{-k})$, and (5) replaced by (7);

$$Y + \ln X \simeq Y_{\text{last}}. \tag{7}$$

Equation (6) will require access to precomputed values of $L_k(d_i)$ for $d_i \in \{0,1,\cdots,r-1\}$ and $1 \leq k < p/2 + 1$. For $k \geq p/2 + 1$, $L_k(d_i) \simeq$ two's complement of $d_i r^{-k}$.

If the logarithmic constants are stored to q bits precision in a read-only memory (ROM), the maximum absolute error in Y_{last} will be

$$|\, e^L \,| < |\ln (1 - 2^{-n})| + p \cdot 2^{-q} < 2^{-n+1} + p \cdot 2^{-q}.$$

C. Algorithm for $Y \cdot \exp (X)$

Following Specker [4], we use the transformation

$$Y \cdot \exp (X) = Y \cdot (\Pi f_i) \cdot [\exp (X - \Sigma \ln f_i)].$$

If

$$X - \Sigma \ln f_i \to 0$$

then

$$Y \cdot \Pi f_i \to Y \cdot \exp (X).$$

Argument range: $0 \leq X < \ln 2$.

The additive normalization of X, and the associated generation of $Y \cdot \exp (X)$, are performed recursively as follows.

Algorithm EXP:

$$X_1 = X, \quad Y_1 = Y, \quad i = 1, \quad k = 1, \quad d_1 = f(X)$$

$$X_{i+1} = X_i - \ln (1 + d_i r^{-k})$$

$$= X_i + L_k(d_i), \quad k \geq 1 \tag{8}$$

$$Y_{i+1} = Y_i (1 + d_i r^{-k}). \tag{9}$$

Assume that X_i has had its $(k-1)$ leading digits liquidated, i.e.,

$$X_i = b_k r^{-k} + b_{k+1} r^{-(k+1)} + \cdots$$

where b_k, b_{k+1}, etc., are equivocal m-bit digits. d_i, $i > 1$ is now selected as

$$d_i = b_k.$$

With this selection of d_i, the kth digit of X_{i+1}, after performing (8), may be 0 or 1, the latter condition being called overflow. Overflow is possible because of the second term in the expansion

$$-\ln (1 + d_i r^{-k}) = -d_i r^{-k} + (d_i^2/2) r^{-2k} - \cdots$$

and is corrected by a repetition on that value of k. As with Algorithm DIV, d_1 must be chosen by table lookup. When $k = p + 1$,

$$Y \cdot \exp (X) \simeq Y_{\text{last}}$$

where the maximum absolute error in $Y_{\text{last}} < 2^{-n+1}$. After 4000 simulated examples, the number of overflows were less than the number of undershoots occurring in the generation of Y/X.

To generate $Y \cdot \exp (-U)$, $0 < U \leq \ln 2$, we use

$$Y \cdot \exp (-U) = \tfrac{1}{2} Y \cdot \exp (\ln 2 - U)$$

$$= \tfrac{1}{2} Y \cdot \exp (X), \quad 0 \leq X < \ln 2.$$

D. Algorithm for Square Root

Following Chen [6], we use the transformation:

$$Y/X^{1/2} = \frac{Y \cdot \Pi f_i}{(X \cdot \Pi f_i^2)^{1/2}}.$$

If

$$X\Pi f_i^2 \to 1,$$

then

$$Y \cdot \Pi f_i \to Y/X^{1/2}.$$

Argument range: $\frac{1}{4} \leq X < 1$. $X^{1/2}$ is computed by setting $Y = X$. The generation of $Y/X^{1/2}$ is performed recursively as

$$X_1 = X, \quad Y_1 = Y$$

$$\hat{X}_{i+1} = X_i (1 + d_i r^{-k})$$

$$X_{i+1} = \hat{X}_{i+1} (1 + d_i r^{-k}), \quad Y_{i+1} = Y_i (1 + d_i r^{-k})$$

with

$$Y/X^{1/2} \simeq Y_{\text{last}}.$$

Assume that X_i has $k-2$ mature leading digits and that its $(k-1)$st digit is at least $r-2$, i.e.,

$$X_i = 0.111\cdots11 \quad \cdots \quad 111\cdots11 \quad 111\cdots1\boxed{x \quad xxx\cdots}x \quad x\cdots \tag{10}$$
$$= 1 - 2r^{-(k-1)} + b_{k-1}r^{-(k-1)} + e_k r^{-k} + e_{k+1}r^{-(k+1)} + \cdots$$

where b_{k-1} is an equivocal bit and e_k, e_{k+1}, etc., are equivocal m-bit digits. d_i is now selected as

$$d_i = \left(b_{k-1}\cdot\frac{r}{2} + \left\lceil\frac{e_k}{2}\right\rceil\right)^*, \qquad i > 1$$

where $\lceil Z\rceil$ means integer part of Z and $(\cdot)^*$ denotes the one's complement of $(\cdot)$. d_i is the one's complement of the bits encompassed by the dashed rectangle in (10). With d_i thus selected, the kth digit of X_{i+1}, $r \geq 8$, will have a value between $r-5$ and $r-1$. If its value is less than $r-2$, undershoot has occurred and is corrected by a recursion on that same value of k.

The maximum absolute error in $Y_{\text{last}} < 2^{-n+1}$.

After 4000 simulated examples, it was estimated that the generation of p digits of $YX^{1/2}$ will take, on the average, less than $p + 0.7$ iterations.

E. Algorithm for $Y\cdot\sin(X)$, $Y\cdot\cos(X)$ [*and* $\tan(X)$]

Following Specker, we use the transformation:

$$\exp(jX) = \beta\cdot\Pi(1 + jd_ir^{-k})\cdot\exp\{j[X - \Sigma\tan^{-1}(d_ir^{-k})]\}$$

where

$$\beta = \Pi\cdot(1 + d_i^2r^{-2k})^{-1/2}$$
$$= \exp\{-\tfrac{1}{2}\Sigma\ln(1 + d_i^2r^{-2i})\}.$$

If

$$X - \Sigma\tan^{-1}(d_ir^{-k}) \to 0,$$

then

$$\text{Im}\{\beta\cdot\Pi(1 + jd_ir^{-k})\} \to \sin X$$

and

$$\text{Re}\{\beta\cdot\Pi(1 + jd_ir^{-k})\} \to \cos X$$

where Im $\{\cdot\}$ denotes imaginary part of $\{\cdot\}$ and Re $\{\cdot\}$ denotes real part of $\{\cdot\}$. Argument range: $0 \leq X \leq \tan^{-1}(1)$.

Assume that X_i has the form

$$X_i = b_kr^{-k} + b_{k+1}r^{-k+1} + \cdots.$$

The generation of $Y\cdot\sin(X)$, $Y\cdot\cos(X)$, or $\tan(X)$ may then be described in the following steps.

Algorithm SC:

1) $U_1 = Y$, $V_1 = Y$, $X_1 = X$, $W_1 = 0$,
   $d_1 = f(X)$.
Loop 2) if $(k = p + 1)$ go to Step 9a) etc.
3) $X_{i+1} = X_i - \tan^{-1}(d_ir^{-k})$.
3a) $U_{i+1} = U_i - V_id_ir^{-k}$.
3b) $V_{i+1} = V_i + U_id_ir^{-k}$.
3c) $W_{i+1} = W_i - \frac{1}{2}\ln(1 + d_i^2r^{-2k})$.
4) If $(X_{i+1} \geq r^{-k})$ go to Step 6).
5) $k = k + 1$.
6) $i = i + 1$.
7) $d_i = b_k$.
8) Go to Step 2).
9a) $\tan(X) \simeq V_{\text{last}}/U_{\text{last}}$.
9b) $Y\cdot\sin(X) \simeq V_{\text{last}}\cdot\exp(-W_{\text{last}})$.
9c) $Y\cdot\cos(X) \simeq U_{\text{last}}\cdot\exp(-W_{\text{last}})$.

Step 3c) will require fast access to the precomputed constants $\frac{1}{2}\ln(1 + d_i^2r^{-2k})$ and Steps 9b) and 9c) will require an invoking of the $Y\cdot\exp(X)$ algorithm.

Maximum absolute error for $Y\cdot\cos(X)$ or $Y\cdot\sin(X) < 2^{-n+2}$. Step 3c), together with Steps 9b) and 9c), results in a faster algorithm than the normal one which divides U_{last} or V_{last} by $(U^2_{\text{last}} + V^2_{\text{last}})^{1/2}$. The number of overflows [Step 4)] in generating U_{last} and V_{last}, by virtue of the series expansion of $\tan^{-1}(d_ir^{-k})$, will be less than those occurring in Algorithm EXP.

IV. DISCUSSION

The radix 2^m implementation of the Y/X, $Y/X^{1/2}$ or $\sin/\cos X$ algorithms will be most efficient when two m BM's, together with variable shifting networks are used in parallel. Observe that the index k occurring in the algorithms would be stored and incremented in a physically existing counter. In practice, the convergent X_i may be left-shifted by m bits every time k is incremented so that the selection of d_i may remain dependent on the same register positions (cf., [5] and [7]).

The $Y\cdot\exp(X)$, $\sin/\cos X$ and $Y + \ln X$ algorithms will require an extra parallel adder and ROM's containing the prestored constants $L_k(d_i)$, $-\tan^{-1}(d_ir^{-k})$ and $-\frac{1}{2}\ln(1 + d_i^2r^{-2k})$. Algorithm SC will require yet another adder to form the W_i.

The error bounds quoted above do not take into account rounding errors which occur during the iterations on i. In order to obtain an answer with p correct digits in radix 2^m, an extra $\log_2((p+1)\cdot m)$ guard digits ought to be used and k should be taken to $p + 2$ with the final result rounded to p digits.

ACKNOWLEDGMENT

The author wishes to thank his colleagues for their assistance during the preparation of this paper, and the referees for their helpful suggestions, one of which was the title of this correspondence.

REFERENCES

[1] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, Sept. 1959.
[2] J. S. Walther, "A unified algorithm for elementary functions," in *1971 Spring Joint Comput. Conf. AFIPS Conf. Proc.*, vol. 38. Montvale, N. J.: AFIPS Press, pp. 379–385.
[3] R. J. Linehardt and H. S. Miller, "Digit by digit transcendental function computation," *RCA Rev.*, vol. 30, pp. 209–247, June 1969.
[4] W. H. Specker, "A class of algorithms for ln X, exp x, sin x, cos x, $\tan^{-1}x$ and $\cot^{-1}x$," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 85–86, Feb. 1965.
[5] B. G. deLugish, "A class of algorithms for automatic evaluation of certain elementary functions in a binary computer," Dep. Comput. Sci., Univ. of Illinois, Rep. 399, June 1, 1970.
[6] T. C. Chen, "Automatic computation of exponentials, logarithms, ratios and square roots," *IBM J. Res. Develop.*, vol. 4, pp. 380–388, 1972.
[7] M. D. Ercegovac, "Radix-16 evaluation of certain elementary functions," *IEEE Trans. Comput.*, vol. C-22, pp. 561–566, June 1973.
[8] S. D. Pezaris, "A 40-ns 17-bit by 17-bit array multiplier," *IEEE Trans. Comput.*, vol. C-20, pp. 442–447, Apr. 1971.
[9] A. Habibi and P. A. Wintz, "Fast multipliers," *IEEE Trans. Comput.*, vol. C-19, pp. 153–157, Feb. 1970.
[10] R. C. Ghest, "A 2's complement digital multiplier, the Am2505," Advanced Microdevices, Appl. Note, Nov. 1971.
[11] E. V. Krishnamurthy, "On optimal iterative schemes for high-speed division," *IEEE Trans. Comput.*, vol. C-19, pp. 227–231, Mar. 1970.
[12] S. F. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M. Powers, "The IBM system/360 model 91, floating point execution unit," *IBM J. Res. Develop.*, vol. 11, pp. 34–53, Jan. 1967.

A Note on Base −2 Arithmetic Logic

C. K. YUEN

***Abstract*—Circuits for performing arithmetic operations using base −2 representations are considered. Study of the counting process leads to a negative binary up–down counter and new simple methods for positive–negative base conversions. The advantage of employing carry–borrow rather than carry–only during additions is pointed out. Certain special features of negation, arithmetic shift, multiplication, and division in base −2 are described.**

Manuscript received January 25, 1974; revised June 3, 1974.
The author is with the Computer Center, Australian National University, Canberra, A.C.T., Australia.