Radix-16 Evaluation of Certain Elementary Functions

MILOŠ D. ERCEGOVAC

Abstract—This paper describes a family of algorithms for evaluation of a class of elementary functions including division, logarithms, and exponentials. The main objective is to demonstrate the feasibility of higher radix implementations, in particular, radix 16, and to compare performance with radix 2. The emphasis is not on optimality of a single algorithm, but rather on the optimality of a class of algorithms. An attempt to implement a much wider class of functions than is presently done in arithmetic units is encouraged by the current level of digital technology and the existence of suitable algorithms. Besides the definitions of the algorithms, which are based on continued products and continued sums, details related to implementation are discussed.

Index Terms—Continued products, continued sums, division, exponential, logarithm, pipelining, radix 16, redundant number, symmetric digit set.

I. Introduction

THE USE of continued products and sums has long been recognized as a practical way to reduce the evaluation of certain elementary functions to a series of additions, shift operations, and perhaps accesses to precomputed constants [1]-[10]. Recently, DeLugish [1] defined efficient algorithms for the evaluation of a wide class of elementary functions based upon continued products and sums, and the use of radix-2 redundant digit set $\{\bar{1}, 0, 1\}$ ($\bar{1} = -1$). The algorithms generate the value of the function on a digit-by-digit basis. Redundancy [13] simplifies the digit selection procedure and accelerates the computation by increasing the probability of a zero digit value. DeLugish has shown that for a class of functions, including division, multiplication, square root, logarithm, exponential, trigonometric, and inverse trigonometric functions, operation times are from one to three multiplication cycle times.

In this paper, we are concerned with speeding up some of the DeLugish algorithms using the radix-16 digit-by-digit evaluation, i.e., 4 bits/step. Only the algorithms for the fractional parts of the radix-2 floating-point numbers are considered. The arguments are assumed to be of $m = \frac{M}{4}$, radix-16 digits precision. By the multiplicative normalization of a given argument $X_0 \in [\frac{1}{2}, 1)$, we mean a sequence of transformations such that $X_0 \cdot \pi_{i=0}^{m} M_i \to 1$ where the multipliers are of the form $M_k = 1 + S_k \cdot 16^{-k}$ for $0 \leqslant k \leqslant m$. Similarly, in the additive normalization, $X_0 - \Sigma_{j=0}^{m} A_i \to 0$ with $A_k = S_k \cdot 16^{-k}$. In both cases, $S_k \in \{\bar{10}, \cdots, 10\}$. As defined by Robertson [12], the set of values for S_k with the redundancy ratio of $\frac{2}{3}$ allows an efficient multiple formation as well as low precision selection rules.

Manuscript received August 11, 1972; revised November 21, 1972. This work was supported in part by NSF Grant GJ813.

The author is with the Department of Computer Science, University of Illinois, Urbana, Ill. 61801.

II. Multiplicative Normalization

The multiplicative normalization is performed recursively as

$$X_{k+1} = X_k(1 + S_k \cdot 16^{-k}), \quad 0 \leqslant k \leqslant m, \tag{2.1}$$

normalizing X_0 to 1 in $m + 1$ steps, by appropriately choosing the values of S_k. S_k can be determined on the basis of X_k, but to keep selection dependent on the same register positions, the scaled remainder is defined as

$$R_k = 16^{k-1}(X_k - 1), \quad 0 < k \leqslant m. \tag{2.2}$$

From (2.1) and (2.2), the scaled remainder recursion follows:

$$R_{k+1} = 16R_k + S_k + S_kR_k \cdot 16^{-k+1}, \quad 0 < k \leqslant m. \tag{2.3}$$

For the chosen symmetric redundant digit set $\{S_k\}$ with $\max\{|S_k|\} = 10$, R_{k+1} will remain in the same bounds as R_k if $-\frac{2}{3} < R_k \leqslant \frac{2}{3}$. Correspondingly, $1 - (\frac{2}{3}) \cdot 16^{-k} < X_k \leqslant 1 + (\frac{2}{3}) \cdot 16^{-k}$. The first step in the derivation of the selection rules, as described in detail in [17], is the calculation of the intervals in the range of R_k for all k and for each value of S_k such that R_{k+1} satisfies the bounds. To preserve continuity of the R_k range, the allowed values for S_k must correspond to the overlapping intervals in the R_k range. These overlaps are caused by the redundancy in the digit set $\{S_k\}$. Since it is possible to choose for the interval boundaries the numbers that are simple in the binary sense, the low precision operations will suffice in the implementation of the selection rules. Since the correspondence between values of S_k and the intervals of R_k reduces, for $2 < k \leqslant m$, to

$$(-2S_k - 1) \leqslant 32R_k \leqslant (-2S_k + 1), \tag{2.4}$$

a simple selection rule will be to form S_k as the scaled remainder rounded to the most significant digit with the opposite sign. Then, it is convenient to perform selection in the first three "irregular" steps using modified rounding instead of using a table lookup or a direct combinational approach.

The following definitions are used in the algorithms.

1) Two's complement representation of scaled remainders:

$$R_k = -r_0 + \sum_{i=1}^{4m} r_i 2^{-i}, \quad r_i \in \{0, 1\} \text{ for all } i. \tag{2.5}$$

2) Truncated scaled remainder:

$$\hat{R}_k = -r_0 + \sum_{i=1}^{6} r_i 2^{-i}. \tag{2.6}$$

3) Nonsign part of $\hat{R}_k$:

$$T_k = \begin{cases} \sum_{i=1}^{6} r_i 2^{-i}, & \text{if } r_0 = 0 \\ \sum_{i=1}^{6} \bar{r}_i 2^{-i}, & \text{if } r_0 = 1. \end{cases} \quad (2.7)$$

4) Step-dependent rounding constant:

$$U_k = \sum_{i=1}^{6} u_i 2^{-i}, \quad u_i \in \{0, 1\} \text{ and } u_i = f_i(\hat{R}_k). \quad (2.8)$$

5) S_k is in sign and magnitude representation.

6) $[Z]$ denotes the integer part of Z. (2.9)

7) $[Z]$ denotes the smallest integer not smaller than Z. (2.10)

Algorithm N (Multiplicative Normalization) (2.11)

Step N1: (Initialize) $k \leftarrow 0$,
$S_0 \leftarrow 1$ if $\frac{1}{2} \leqslant X_0 < \frac{5}{8}$,
$S_0 \leftarrow 0$ if $\frac{5}{8} \leqslant X_0 < 1$,
$R_1 \leftarrow X_0(1 + S_0) - 1$.

Step N2: (Loop) for $0 < k \leqslant m$:
$k \leftarrow k + 1$,
$|S_k| \leftarrow [(T_k + U_k)16]$; sign $(S_k) \leftarrow \overline{\text{sign}}\,(R_k)$,
if $k < k_1$, then
$R_{k+1} \leftarrow 16R_k + S_k + S_k R_k \cdot 16^{-k+1}$;
otherwise $R_{k+1} \leftarrow 16R_k + S_k$

where

a) $U_k = \sum_{i=1}^{6} u_i 2^{-i}$, $u_1 = u_2 = 0$,
$u_3 = K_1 r_0 \bar{r}_2$,
$u_4 = K_1 r_0 \bar{r}_4 (\bar{r}_2 + \bar{r}_3)$,
$u_5 = K_1(r_0 + \bar{r}_3 \bar{r}_4) + K_2[r_0 + \bar{r}_1(\bar{r}_2 + \bar{r}_3) + r_6] + K$,
$u_6 = K_1 \bar{r}_3 r_4 + K_2 r_0 (r_1 + r_2 r_3)$,

and K_1, K_2, and K denote steps $k = 1$, $k = 2$, and $k > 2$. respectively.

b) $k_1 = [(m + 3)/2]$.

This simplification is the consequence of the decreasing effect of the term $S_k R_k 16^{-k+1}$ on R_{k+1}. It is a useful feature of this normalization procedure that at every step, an increasing number of digits S_k can be obtained from R_k. This reveals the possibility of a variable radix approach to achieve higher speed. At the step $k = k_1$, all remaining digits $S_k, S_{k+1}, \cdots, S_m$ are available after replacing the leading digits of R_{k_1} with the digits from the set $\{\bar{8}, \cdots, 8\}$, in such a way that the value of the truncated remainder remains unchanged. This also reduces the error bound of the normalized argument [14] to

$$|\epsilon_{m+1}| = |1 - X_{m+1}| \leqslant (\tfrac{8}{15})\, 16^{-m}. \quad (2.12)$$

III. Additive Normalization

The additive normalization is a right-directed recoding where one replaces the nonredundant digit set $\{0, \cdots, 15\}$ with a redundant one $\{\bar{10}, \cdots, 10\}$. The procedure is simple and exact. Rounding, as a selection rule, applies to all steps. Since

$$X_{k+1} = X_k - S_k 16^{-k}, \quad 0 \leqslant k \leqslant m, \quad (3.1)$$

with the scaled remainder defined as $R_k = 16^{k-1} X_k$, the basic scaled remainder recursion is

$$R_{k+1} = 16R_k - S_k, \quad 0 < k \leqslant m \quad (3.2)$$

and $|R_k| \leqslant \frac{2}{3}$.

Algorithm A (Additive Normalization) (3.3)

Step A1: (Initialize) $k \leftarrow 0$,
$S_0 \leftarrow 1$,
$R_1 \leftarrow X_0 - S_0$.

Step A2: (Loop) for $0 < k \leqslant m$:
$k \leftarrow k + 1$,
$|S_k| \leftarrow [(T_k + U_k)16]$; sign $(S_k) =$ sign (R_k),
$R_{k+1} \leftarrow 16R_k - S_k$

where $U_k = \frac{1}{32}$. This choice of the rounding constant restricts S_k to $\{\bar{8}, \cdots, 8\}$, which is sufficient for recoding.

IV. Division

Consider

$$Q = \frac{Y_0}{X_0} = \frac{Y_0 \prod_{i=0}^{m} M_i}{X_0 \prod_{i=0}^{m} M_i} \quad (4.1)$$

where $M_k = 1 + S_k \cdot 16^{-k}$, $0 \leqslant k \leqslant m$. If the digits S_k are determined using the multiplicative normalization (Algorithm *N*), then

$$X_0 \prod_{i=0}^{m} M_i = X_{m+1} = 1 + \epsilon_{m+1} \quad (4.2)$$

where $|\epsilon_{m+1}| \leqslant (\frac{8}{15}) \cdot 16^{-m}$, given in (2.12). Consequently,

$$Y_0 \prod_{i=0}^{m} M_i = Q_{m+1} \approx Q \quad (4.3)$$

with the relative error $|\delta| = |\epsilon_{m+1}|$. The partial results are defined recursively as

$$Q_{k+1} = Q_k(1 + S_k 16^{-k}), \quad 0 \leqslant k \leqslant m$$
$$Q_0 = Y_0. \quad (4.4)$$

An example of the division algorithm is given in Fig. 1. The multiplicative normalization scheme is shown in Fig. 2, and the result evaluation scheme is shown in Fig. 3.

Algorithm D (Division) (4.5)

	Normalization:	Result Evaluation:
Step D1: $k \leftarrow 0$;		$Q_0 \leftarrow Y_0$;
Step D2: for $k \leqslant m$:	Algorithm *N*	$Q_{k+1} \leftarrow Q_k + Q_k S_k 16^{-k}$.

Note in Fig. 1 that the six leading digits of R_7 represent, after recoding, the remaining digits $S_7, \cdots, S_{12}$: $0.597\text{BEA}9 \cdots = 0.6\bar{7}8\bar{4}\bar{1}\bar{5} \cdots$.

X0= 0.70999997854232 Y0= 0.59314718055994 Y0/X0= 0.83541858941702

K	SK	RK+1 (IN HEXADECIMAL)	XK+1 (IN DECIMAL)	QK+1 (IN DECIMAL)
0	0	-.4A3D7100000238	0.70999997854232	0.59314718055994
1	7	-.547AD8FFFFC000	1.02062496915459	0.85264907205491
2	-5	-.20472C2FFC0FF0	1.00059088772578	0.83599576986634
3	-3	-.2C1512B4C1F79F	0.99995795982950	0.83538346827708
4	3	-.3EA693C067A181	1.00000373427224	0.83542170909747
5	-4	-.1597BE93D4E68E	0.99999991956073	0.83541852221656
6	1	-.597BEA96CA521D	0.99999997916537	0.83541857201138
7	6	.6841547A735EA7	1.00000000151711	0.83541859063644
8	-7	-.78EAB88666AA85	0.99999999988730	0.83541858932287
9	8	.41547795B601EB	1.00000000000371	0.83541858942012
10	-4	.15477958FC992	1.00000000000008	0.83541858941708
11	-1	.547795B4FC83D8	1.00000000000002	0.83541858941704
12	-5	.47795B4FC8231A	1.00000000000000	0.83541858941702

Fig. 1. Example of the division algorithm.

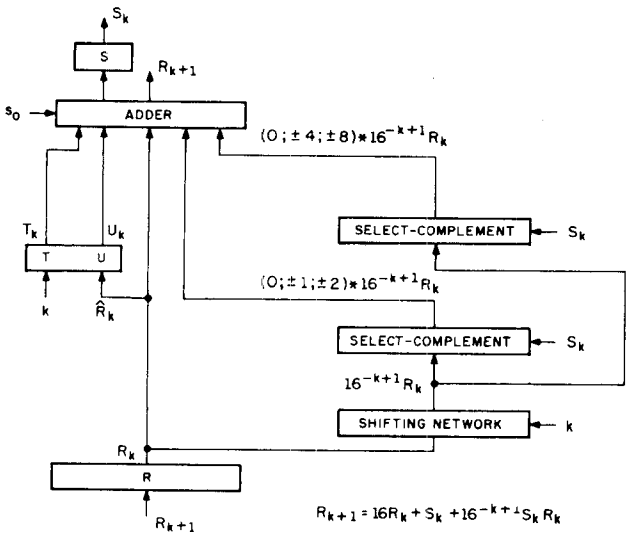


Fig. 2. Multiplicative normalization scheme.

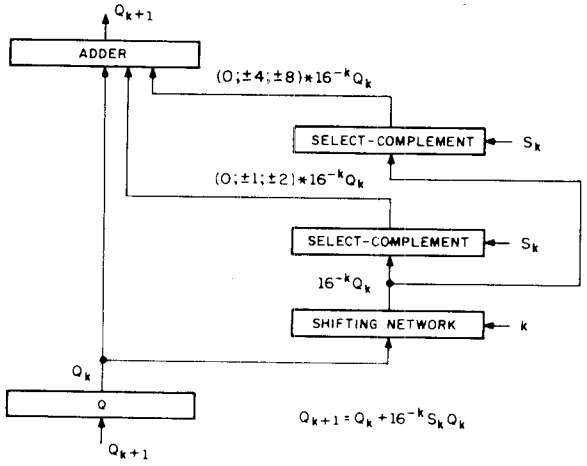


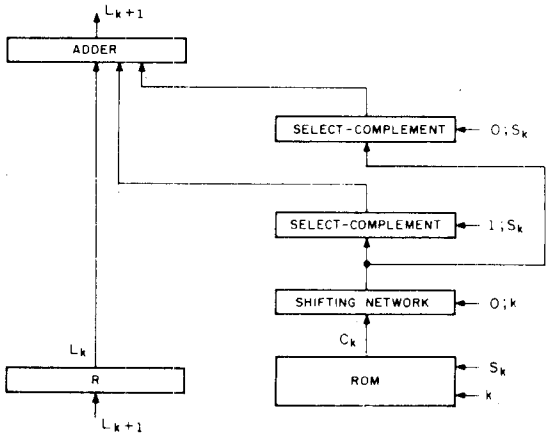
Fig. 3. Result evaluation scheme.

V. Logarithm

For a floating-point argument $X = X_0 2^{E_x}$

$$\ln X = \ln X_0 + E_x \ln 2, \tag{5.1}$$

but we consider only an algorithm for $\ln X_0$, the calculation of the second term presenting no problems.



$$L_{k+1} = L_k + C_k, 0 \le k < k_1$$

$$C_k = \begin{cases} -\log_{16}(M_k), & 0 \le k < k_1 \\ 1, & k_1 \le k \le m \\ \log_{16} 2, & \text{for Ex } \log_{16} 2 \end{cases} \qquad L_{k+1} = L_k - S_k 16^{-k},\ k_1 \le k \le m$$

Fig. 4. Result evaluation scheme with ROM.

$$\ln X_0 = \ln \left(X_0 \prod_{i=0}^{m} M_i \right) - \ln \left(\prod_{i=0}^{m} M_i \right) \tag{5.2}$$

with $M_k = 1 + S_k \cdot 16^{-k}$, $0 \le k \le m$. According to the Algorithm N, $|1 - X_0 \pi_{i=0}^{m} M_i| \le (\frac{8}{15}) \cdot 16^{-m}$. Then

$$\ln X_0 \approx -\sum_{i=0}^{m} \ln (1 + S_i \cdot 16^{-i}). \tag{5.3}$$

Since the logarithmic constants $\ln (1 + S_k \cdot 16^{-k})$ are stored with m-digits precision in a fast Read-Only memory (ROM), the result will contain an accumulated roundoff error $|\epsilon_R| \le (\frac{1}{2})m \cdot 16^{-m}$, inversely proportional to the radix of implementation. Due to the finite precision, the actual requirement for the ROM capacity is decreased by approximately one-half, as follows from the power series expansion for the logarithm:

$$\ln (1 + S_k \cdot 16^{-k}) \approx S_k \cdot 16^{-k} \tag{5.4}$$

for $k \ge k_1$ where $k_1 = \lceil (2 \log_2 10 - 1 + 4m)/8 \rceil$.

See the multiplicative normalization scheme in Fig. 2, and the result evaluation scheme with ROM in Fig. 4.

Algorithm L (Logarithm) (5.5)

Normalization:	Result Evaluation:
Step L1: $k \leftarrow 0$;	$L_0 \leftarrow 0$;
Step L2: for $k \le m$: [Algorithm N]	if $k < k_1$, then $L_{k+1} \leftarrow L_k - \ln(1 + S_k 16^{-k})$; otherwise $L_{k+1} \leftarrow L_k - S_k 16^{-k}$.

VI. Exponential

An initial manipulation [3], [1] of the argument reduces the evaluation of e^X to the evaluation of e^{X_0} with $|X_0| < \ln 2$. Let $X \log_2 e = I + F$ where I and F denote the integer and fractional parts, respectively. Since $X = X \log_2 e \ln 2$,

$$e^X = 2^I e^{F \ln 2} = 2^I e^{X_0}. \tag{6.1}$$

The factor 2^I is easily incorporated into the exponent part of the result. To simplify the initial step, F is restricted to $F < 0$ and I is correspondingly adjusted so that $X_0 \in (-\ln 2, 0]$.[1] Consider

$$X_0 = X_0 - \ln\left(\prod_{i=0}^{m} M_i\right) + \ln\left(\prod_{i=0}^{m} M_i\right) \tag{6.2}$$

where $M_k = 1 + S_k \cdot 16^{-k}, 0 \leqslant k \leqslant m$. If $X_0 - \ln(\pi_{i=0}^{m} M_i) \approx 0$, then $e^{X_0} \approx \pi_{i=0}^{m} M_i$. The corresponding result evaluation recursion is

$$Q_0 = 1$$

$$Q_{k+1} = Q_k(1 + S_k \cdot 16^{-k}), \quad 0 \leqslant k \leqslant m. \tag{6.3}$$

The constants S_k are determined by the additive normalization, using the scaled remainder recursion:

$$R_{k+1} = 16R_k - 16^k \ln(1 + S_k \cdot 16^{-k}), \quad 0 < k \leqslant m. \tag{6.4}$$

The set of stored constants is the same as the one for the logarithmic algorithm. For $k \geqslant k_1$, (5.4), the normalization is defined by the Algorithm A, Step $A2$. As derived in [17], the rounding with $U_k = \frac{1}{32}$ is the selection procedure in all steps, except in the initial, which is performed according to the following table.

X_0	M_0	$\ln M_0$
$[-\frac{1}{8}, 0]$	1	0
$[-\frac{3}{8}, -\frac{1}{8})$	$e^{-1/4}$	$-\frac{1}{4}$
$(-\ln 2, -\frac{3}{8})$	$e^{-17/32}$	$-\frac{17}{32}$

The constants, necessary to be precomputed, are stored in the ROM. A summary of the procedure for calculation of e^X follows.

Preparatory Operations P (6.5)

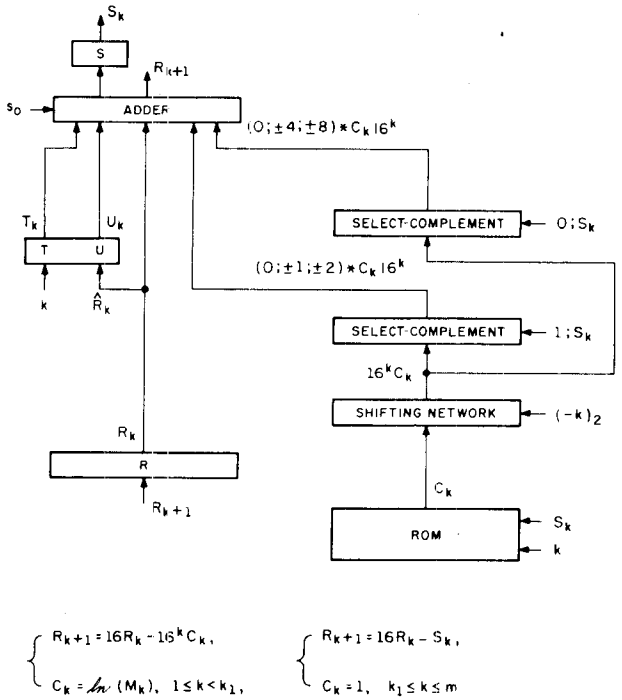
Step P1: $N \leftarrow X \log_2 e$.
Step P2: If $X > 0$, then $I \leftarrow [N] + 1$; otherwise $I \leftarrow [N]$.
Step P3: $X_0 \leftarrow (N - I) \ln 2$.

The additive normalization scheme with ROM is shown in Fig. 5, and the result evaluation scheme is shown in Fig. 2.

Algorithm E (Exponential) (6.6)

	Normalization:	Result Evaluation:
Step E1: $k \leftarrow 0$;	$R_1 \leftarrow X_0 - \ln M_0$;	$Q_1 \leftarrow M_0$;
Step E2: for $k \leqslant m$:	if $k < k_1$, then $\lvert S_k \rvert \leftarrow [(T_k + U_k)16]$; sign $(S_k) \leftarrow$ sign (R_k); $R_{k+1} \leftarrow 16R_k - 16^k \ln(1 + S_k 16^{-k})$; otherwise Step $A2$;	$Q_{k+1} \leftarrow Q_k + Q_k S_k 16^{-k}$;

[1] Two multiplications in the X_0 calculation could be replaced by one division of X by ln 2 so that $X = I \ln 2 + X_0$ if the division scheme gives the remainder [2], [11].



$$\begin{cases} R_{k+1} = 16R_k - 16^k C_k, \\ C_k = \ln(M_k), \quad 1 \leq k < k_1, \end{cases} \qquad \begin{cases} R_{k+1} = 16R_k - S_k, \\ C_k = 1, \quad k_1 \leq k \leq m \end{cases}$$

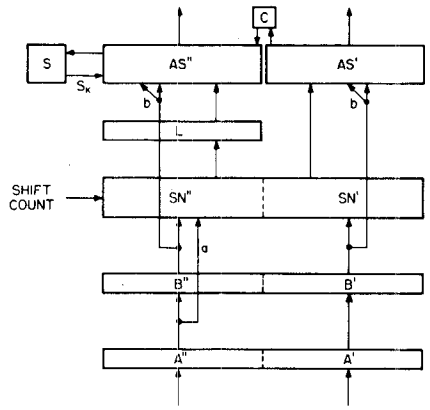
Fig. 5. Additive normalization scheme with ROM.

where k_1 is defined in (5.4) and $U_k = \frac{1}{32}$.

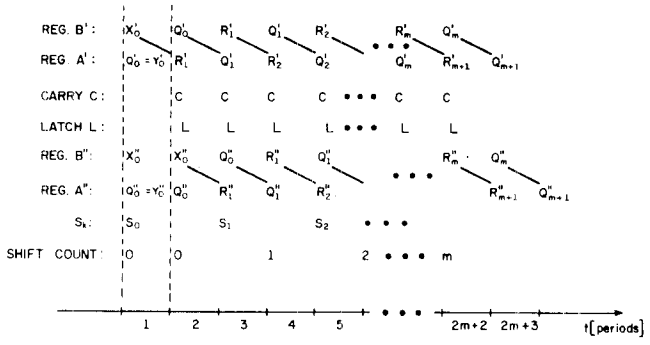
VII. Implementation

This investigation of the use of radix 16 in the implementation of the described algorithms has been motivated by the speed improvement over the radix-2 approach and by possible tradeoffs in the hardware requirements. A brief discussion of the radix-16 implementation and some basic comparisons with the radix-2 approach [1] follow, with more details given in [17]. The general configuration may contain two arithmetic units, as the fastest solution, or the normalization and the result evaluation can be performed in an overlapped mode, using one "pipelined" arithmetic unit. In either case, the main parts of the arithmetic unit are as follows. The three-input adder with the multiple formation networks is estimated to be twice as complex in the radix-16 case as the corresponding part in the radix-2 case. Speedwise, we estimate that the add times satisfy $t_{a_{16}} < 1.2 t_{a_2}$. The fast variable shifting network of the "barrel switch"-type [16], on the other hand, requires more than 30 percent more hardware in the radix-2 case. Speedwise, $t_{sh_2} > 1.3 t_{sh_{16}}$, where t_{sh} denotes the shifting delay. The selection procedure in the radix-2 case requires a simple 4-bit comparison. The radix-16 approach requires 7 bits of precision for selection, implementation of the 5 $u_i = f(\hat{R}_k)$ equations (2.11), costing less than 40 literals, as well as the additional inputs to the 7 most significant positions of the adder. In both cases, the selection requirements are negligible in comparison with the rest of the unit. The control is also simple in both cases.

From the above considerations, the hardware requirement ratio for the radix 2 and the radix 16 is approximately 2:3. The basic cycle can be taken to be the same, since the add time is dominant over control, selection, and shifting time.



(a)



(b)

Fig. 6. (a) Pipelined organization. (b) Timing chart for the division algorithm.

The ROM capacity requirement ratio is about 1:3 in favor of the binary case.

Let the performance of an implementation in the radix r be $P_r = \log_2 r/T_r$, where T_r is the total delay necessary to evaluate $\log_2 r$ bits of the result, as defined in [15]. T_r is equivalent to the basic cycle. In the radix-2 case with $S_k \in \{\bar{1}, 0, 1\}$, the probability of $S_k = 0$ ($p_0 = \frac{2}{3}$) is utilized by providing an adder bypass and reducing the number of full basic cycles to $M/3$ on the average, where M is the number of bits. Then it can be taken that the radix-16 basic cycle is $T_{16} \cong 3T_2$, since the number of basic cycles in the radix-16 case is always the same, the probability of $S_k = 0$ being too low. Then the ratio of performances is $P_{16}/P_2 \cong \frac{4}{3}$ on the average. If the efficiency of the implementation is defined as the ratio between performance and cost per bit, then with all previous assumptions, $E_{16}/E_2 \cong 1$ without considering ROM requirements. If the ROM capacity requirement is taken into account, then the radix-2 approach will offer more efficient design, but the radix-16 case will maintain better performance with shorter execution time. The selection procedure for radix 16 has been shown to be sufficiently simple.

One way to overlap the normalization and the result evaluation using pipelining techniques is shown in Fig. 6. The timing chart (b) corresponds to the division algorithm. The superscripts $'$ and $''$ denote the right and the left parts of all networks, registers, and corresponding results. Two periods now define the basic cycle, somewhat longer so that the pipelined mode is 15–25 percent slower than the two separate arithmetic unit mode. The adder with multiple formation networks is split into two equal parts, AS'' and AS', by breaking the carry path and inserting a one-bit carry register C. Outputs from the left half SN'' of the shifting network are to be saved in a latch L.[2] The selection is carried out in the block S on the basis of adder outputs and returns the value of S_k. The initial operands are in register B for normalization, and in register A for result evaluation. Each register contains two separately controlled halves: B'', B' and A'', A'. The outputs from AS'' and AS' are connected, under a separate control, to the inputs of A'' and A' registers, respectively. A more detailed description can be found in [17].

VIII. Conclusions

The use of a higher radix in implementations makes faster execution of the algorithms possible, but feasibility of the corresponding selection procedure requires some consideration. For the radix-16 case, we have shown that the selection rules are very simple for all except the initial three steps, in which case they are of an acceptable complexity. Furthermore, in the exponential algorithm, only the initial step requires a specialized procedure. With the current trend in lower hardware cost, the corresponding realization requirements can be easily justified. It is believed that similar selection procedures could be applied for square-root, trigonometric, and inverse trigonometric function evaluation in radix 16. Multiplication has been made compatible [17].

[2]Two additional 4-bit latches, not shown here, are needed to hold bits shifted left from AS' and SN'.

ACKNOWLEDGMENT

The author wishes to thank Prof. J. E. Robertson of the University of Illinois, Urbana, for highly valued guidance and discussions. Thanks are also due to the referees for helpful comments.

REFERENCES

[1] B. G. DeLugish, "A class of algorithms for automatic evaluation of certain elementary functions in a binary computer," Ph.D. dissertation, Dep. Comput. Sci., Univ. Illinois, Urbana, June 1970.
[2] H. Marx, "Additionsverfahren zur Berechnung des Logarithmus und der Exponentialfunktion (I)," *Mitt. aus dem Mathem. Seminar Giessen*, no. 54, Giessen 1956.
[3] E. G. Kogbetliantz, "Computation of e^N for $-\infty < N < +\infty$ using an electronic computer," *IBM J. Res. Develop.*, vol. 1, pp. 110–115, Apr. 1957.
[4] J. E. Volder, "The CORDIC trigonometric computing technique," *IEEE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, Sept. 1959.
[5] I. Y. Akushsky *et al.*, "Methods of speeding up the operation of digital computers," in *Proc. ICIP*, June 1959, pp. 382–389.
[6] J. E. Meggitt, "Pseudo division and pseudo multiplication processes," *IBM J. Res. Develop.*, vol. 6, pp. 210–226, Apr. 1962.
[7] 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.* (Short Notes), vol. EC-14, pp. 85–86, Feb. 1965.
[8] 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, 1971, pp. 379–385.
[9] T. C. Chen, "Automatic computation of exponentials, logarithms, ratios and square roots," *IBM J. Res. Develop.*, vol. 16, pp. 380–388, July 1972.
[10] C. V. Ramamoorthy, J. R. Goodman, and K. H. Kim, "Some properties of iterative square-rooting methods using high-speed multiplication," *IEEE Trans. Comput.*, vol. C-21, pp. 837–847, Aug. 1972.
[11] H. J. Maehly, "Rational approximations for transcendental functions," in *Proc. ICIP*, June 1959, pp. 57–62.
[12] J. E. Robertson, "A new class of digital division methods," *IRE Trans. Electron. Comput.*, vol. EC-7, pp. 218–222, Sept. 1958.
[13] ——, lecture notes for computer science courses 394 and 482, Dep. Comput. Sci., Univ. Illinois, Urbana, Fall 1970 and Spring 1971.
[14] ——, private communication, Sept. 1972.
[15] D. E. Atkins, "A study of methods for selection of quotient digits during digital division," Ph.D. dissertation, Dep. Comput. Sci., Univ. Illinois, Urbana, June 1970.
[16] R. L. Davis, "The ILLIAC IV processing element," *IEEE Trans. Comput.*, vol. C-18, pp. 800–816, Sept. 1969.
[17] M. D. Ercegovac, "Radix 16 division, multiplication, logarithmic and exponential algorithms based on continued product representations," M.Sc. thesis, Dep. Comput. Sci., Univ. Illinois, Urbana, Aug. 1972.

Miloš D. Ercegovac was born in Belgrade, Yugoslavia, on June 28, 1942. He received the diploma in electrical engineering from the University of Belgrade, Belgrade, Yugoslavia, in 1965, and the M.S. degree in computer science from the University of Illinois, Urbana, in 1972.

From February 1966 to June 1970 he was with the Institute for Automation and Telecommunications "Mihailo Pupin," Belgrade, Yugoslavia, where he participated in the design and development of digital computers. Since July 1970 he has been a Research Assistant in the Department of Computer Science, University of Illinois, Urbana. His research interests include computer arithmetic, logical design, and computing systems organization. He is currently a Ph.D. student in computer science at the University of Illinois.