Fast Computational Algorithms for Bit Reversal

ROBERT J. POLGE, MEMBER, IEEE, B.K. BHAGAVAN, AND JAMES M. CARSWELL

Abstract—**Radix—2 fast Fourier transform programs of the Gentleman-Sande type leave the transformed array in a scrambled order of frequencies. Unscrambling is accomplished by moving each element from its present location into a new location obtained by bit reversal. Bit reversal can be interpreted as an exchange of groups of bits symmetric with respect to a pivot. Two formulas are developed for simultaneous and sequential exchanging in place, with negligible auxiliary storage. Programs for one-step and sequential unscrambling were implemented. Binary unscrambling is useful to unscramble large arrays in peripheral storage. The one-step unscrambling program presented is more efficient than other programs available in the literature.**

Index Terms—**Bit reversal, fast Fourier transform, unscrambling.**

I. Introduction

THE fast Fourier transform (FFT) programs based on the Gentleman-Sande algorithm [1] leave the transformed array in a scrambled order of frequencies. The unscrambling for a radix—2 transformation is achieved by bit reversal where the new address of an element is obtained by reversing the order of the binary expansion of the original address. The implementation of bit reversal on a general-purpose computer is important since unscrambling accounts for a significant portion of the total FFT computation time.

In this paper two formulas for unscrambling are developed by interpreting bit reversal as an exchange of symmetric bits. The first formula shows that a complex array of size $N = 2^M$ can be unscrambled directly given an integer array of size $\sqrt{N}$ if M is even or $\sqrt{N/2}$ if M is odd. The second formula is more general; it shows that unscrambling can be performed in steps where each step corresponds to the exchange of groups of symmetric bits. Two applications of the formulas, the one-step algorithm and the binary-pair algorithm, have been implemented on a computer and listings of the programs are included for reference. The binary-pair algorithm has been adapted to unscramble arrays exceeding core capacity. The binary-pair and the one-step algorithms are compared with two others suggested in [1] and [2] for computing efficiency and compactness of coding. All the algorithms were coded in Fortran and the computations were carried out on the Univac-1108 Executive 8 system.

Manuscript received October 19, 1972; revised April 19, 1973. This work was supported in part by the Advanced Ballistic Defense Missile Agency, U.S. Army, Huntsville, Ala.

R.J. Polge is with the Department of Electrical Engineering, University of Alabama, Hunstville, Ala.

B.K. Bhagavan is with the Research Institute, University of Alabama, Huntsville, Ala.

J.M. Carswell was with the University of Alabama, Huntsville, Ala. He is now with North American Aviation, Downey, Calif.

II. The FFT and the Need for Bit Reversal

The Fourier transform and the Fourier series are approximated on the digital computer by a finite summation called the discrete Fourier transform (DFT). Consider a set X of N complex samples and denote as X' the direct DFT of X. The elements $x'(k)$ are defined by a finite summation

$$x'(k) = \frac{1}{N} \sum_{j=0}^{N-1} w^{-jk} x(j) \tag{1}$$

where the complex coefficient w is equal to $\exp(i2\pi/N)$. Similarly, the inverse DFT is defined as

$$x(j) = \sum_{k=0}^{N-1} w^{jk} x'(k). \tag{2}$$

The DFT is very useful; unfortunately its computation is time consuming. For example, the computation of X from X' requires N^2 complex multiplications and $N(N-1)$ complex summations.

The FFT is an efficient computational algorithm, originated by Cooley and Tukey [3] to compute the DFT. Many algorithms have been developed since. Gentleman and Sande [1] have proposed an FFT algorithm that minimizes computer storage by using the same array for the input data and for the transformed data. However, the transformed samples are not stored at the proper location and the transformation must be followed by a permutation called unscrambling. Only the radix—2 FFT is considered in this paper. In this case N is a power of 2 and the unscrambling corresponds to bit reversal.

The radix—2 FFT algorithm of an array of size $N = 2^M$ is based on the factorization of N into a product of M factors each of which is equal to 2. It is implemented in the following four steps: initialization; a sequence of transformations that corresponds to a partial transformation with respect to each factor; a division by N in the case of a direct transformation; and an unscrambling procedure.

For example, assume that $N = 8$. The factorization of N into a product of factor 2 corresponds to a binary expansion of j and k. Substituting $j = 4j_3 + 2j_2 + j_1$ and $k = 4k_3 + 2k_2 + k_1$ in (2), factoring $w^{(4j_3 + 2j_2 + j_1)(4k_3 + 2k_2 + k_1)}$ and collecting with respect to the j_i variables yields

$$x(j_3 4 + j_2 2 + j_1) = \sum_{k_1=0}^{1} w^{4j_3 k_1} \sum_{k_2=0}^{1} w^{2j_2(2k_2 + k_1)} \cdot \sum_{k_3=0}^{1} w^{j_1(4k_3 + 2k_2 + k_1)} \cdot x'(k_3 4 + k_2 2 + k_1) \tag{3}$$

where j_i and k_i take the values 0 and 1, $w = \exp(i2\pi/8) = \cos(\pi/4) + i \sin(\pi/4)$, and the factors $w^{j_2 k_3 8}$, $w^{j_3 k_2 8}$, and $w^{j_3 k_3 16}$ were deleted because they remain equal to 1 for any combinations of j_2, j_3, k_2, and k_3.

Equation (3) is implemented in the following four steps:

$$x^{(1)}(j_1 4 + k_2 2 + k_1) = w^{j_1(2k_2 + k_1)} \sum_{k_3=0}^{1} w^{4 j_1 k_3} \cdot x'(k_3 4 + k_2 2 + k_1) \quad (4a)$$

$$x^{(2)}(j_1 4 + j_2 2 + k_1) = w^{2 j_2 k_1} \sum_{k_2=0}^{1} w^{4 j_2 k_2} \cdot x^{(1)}(j_1 4 + k_2 2 + k_1) \quad (4b)$$

$$x^{(3)}(j_1 4 + j_2 2 + j_3) = \sum_{k_1=0}^{1} w^{4 j_3 k_1} \cdot x^{(2)}(j_1 4 + j_2 2 + k_1) \quad (4c)$$

$$x(j_3 4 + j_2 2 + j_1) = x^{(3)}(j_1 4 + j_2 2 + j_3) \quad (4d)$$

where (4a) to (4c) are the three partial transformations with respect to the factor 2, and (4d) shows that a permutation is required because the samples are stored at a wrong address where the least significant bits are used in place of the most significant bits.

The array $X^{(3)}$ is said to be scrambled because a transformed sample, instead of being stored at the proper location k, is stored at an address $\hat{k}$ obtained by binary expansion of k and bit reversal. Equation (4d) shows the unscrambling procedure, namely, that $x^{(3)}(\hat{k})$ is stored into $x(k)$, where $k = j_3 4 + j_2 2 + j_1$ and $\hat{k} = j_1 4 + j_2 2 + j_3$. More generally, when $N = 2^M$,

$$k = \sum_{m=1}^{M} j_m 2^{m-1} \text{ and } \hat{k} = \sum_{m=1}^{M} j_{M-1-m} 2^{m-1}. \quad (5)$$

The partial transformations defined by (4a), (4b), and (4c) are performed in place. That is, the arrays X', $X^{(1)}$, $X^{(2)}$, and $X^{(3)}$ can share the same storage, and the superscript will drop subsequently. To continue to preserve storage one must also perform the unscrambling in place without using an auxiliary array. This can be accomplished by exchanging the elements $x(\hat{k})$ and $x(k)$ in the array X. The elements need to be exchanged only when $\hat{k} > k$ (or alternatively when $\hat{k} < k$). Note that if both $\hat{k} > k$ and $\hat{k} < k$ were used, every exchange will be canceled by an identical exchange. Therefore, unscrambling is defined by the sequence of exchanges

$$x(k) \rightleftarrows x(\hat{k}), \quad \text{for } \hat{k} > k. \quad (6)$$

The index $\hat{k}$ could be computed from (5), but this computation requires too many operations. Instead, note that $\hat{k}$ is obtained from k by exchanging symmetric bits. More precisely, the binary expansion of a general address k, $0 \leqslant k < N$ is expressed as $j_M j_{M-1} \cdots j_2 j_1$, where $j_i = 0$ or 1 for $i = 1, 2, \cdots, M$. The pivot of the address is defined to be the central bit if M is odd and to be between two central bits if M is even. Bit reversal corresponds to an exchange of the bits symmetric with respect to the pivot, i.e., an exchange between j_m and j_{M+1-m} where $m \leqslant [\frac{M}{2}]$, with $[\cdot]$ denoting "integer part of." Clearly, the exchange is necessary only when $j_m \neq j_{M+1-m}$. The exchange of pairs of bits can be performed all in one step, or can be done sequentially in groups of one or more pairs at a time. This property will be used to develop efficient unscrambling formulas for unscrambling in one or more steps. Table I shows how $\hat{k}$ is obtained from k by bit exchange and the ordering of X before and after unscrambling.

III. Formula for One-Step Bit Reversal

This section shows that the bit reversal defined by (6) can be implemented in one step by the direct computation of $\hat{k}$ from k. The two cases, M even and M odd, are similar, but do differ sufficiently to justify separate development.

M Even

When M is even, the M bits can be equally divided into two groups of magnitude $M_1 = M/2$. An M-bit number k may be expressed as a function of two M_1-bit numbers k_1 and k_2 by

$$k = k_2 N_1 + k_1 \quad (7)$$

where $k = 0, 1, \cdots, (N-1)$, $N_1 = 2^{M_1} = \sqrt{N}$, and $k_1, k_2 = 0, 1, \cdots, (N_1 - 1)$. Thus, if the binary expansion of k is

$$k = j_M j_{M-1} \cdots j_{M-M_1+1} j_{M-M_1} \cdots j_2 j_1$$

then the binary expansion of k_1 and k_2 are

$$k_1 = j_{M-M_1} \cdots j_2 j_1$$

and

$$k_2 = j_M j_{M-1} \cdots j_{M-M_1+1}.$$

Therefore, the bit-reversed version $\hat{k}$ of k can be written as

$$\begin{aligned} \hat{k} &= j_1 j_2 \cdots j_{M-1} j_M \\ &= (j_1 j_2 \cdots j_{M-M_1}) N_1 + j_{M-M_1+1} \cdots j_{M-1} j_M \\ &= \hat{k}_1 N_1 + \hat{k}_2 \end{aligned} \quad (8)$$

where $\hat{k}_1$ and $\hat{k}_2$ are obtained by bit-reversing k_1 and k_2, respectively. Equations (7) and (8) will be used as a pair

$$\begin{aligned} k &= k_2 N_1 + k_1 \\ \hat{k} &= \hat{k}_1 N_1 + \hat{k}_2. \end{aligned} \quad (9)$$

Equation (8) shows that bit reversal of an M-bit group is equivalent to the following sequence: 1) bit reversal of the M_1 least significant bits, 2) bit reversal of the M_1 most significant bits, and 3) exchange of the most significant and least significant groups of M_1 bits. Therefore, if k is an address for an array of size $N = 2^M$, where M is even, the bit-reversed

TABLE I
EXAMPLES OF BIT REVERSAL BY EXCHANGE OF BITS[a]

A) $M = 4, N = 16$

k	Binary Expansion of k	Binary Expansion of $\hat{k}$	$\hat{k}$	Exchange $x(k) \rightleftharpoons x(\hat{k})$
0	0 0 | 0 0	0 0 | 0 0	0	not needed
1	0 0 | 0 1	1 0 | 0 0	8	$x(1) \rightleftharpoons x(8)$
2	0 0 | 1 0	0 1 | 0 0	4	$x(2) \rightleftharpoons x(4)$
3	0 0 | 1 1	1 1 | 0 0	12	$x(3) \rightleftharpoons x(12)$
4	0 1 | 0 0	0 0 | 1 0	2	see $k = 2$
5	0 1 | 0 1	1 0 | 1 0	10	$x(5) \rightleftharpoons x(10)$
6	0 1 | 1 0	0 1 | 1 0	6	not needed
7	0 1 | 1 1	1 1 | 1 0	14	$x(7) \rightleftharpoons x(14)$
8	1 0 | 0 0	0 0 | 0 1	1	see $k = 1$
9	1 0 | 0 1	1 0 | 0 1	9	not needed
10	1 0 | 1 0	0 1 | 0 1	5	see $k = 5$
11	1 0 | 1 1	1 1 | 0 1	13	$x(11) \rightleftharpoons x(13)$
12	1 1 | 0 0	0 0 | 1 1	3	see $k = 3$
13	1 1 | 0 1	1 0 | 1 1	11	see $k = 11$
14	1 1 | 1 0	0 1 | 1 1	7	see $k = 7$
15	1 1 | 1 1	1 1 | 1 1	15	not needed

B) $M = 5, N = 32$ (this table is incomplete)

0	0 0 0 | 0 0	0 0 0 | 0 0	0	not needed
1	0 0 0 | 0 1	1 0 0 | 0 0	16	$x(1) \rightleftharpoons x(16)$
2	0 0 0 | 1 0	0 1 0 | 0 0	8	$x(2) \rightleftharpoons x(8)$
3	0 0 0 | 1 1	1 1 0 | 0 0	24	$x(3) \rightleftharpoons x(24)$
10	0 1 0 | 1 0	0 1 0 | 1 0	10	not needed
11	0 1 0 | 1 1	1 1 0 | 1 0	26	$x(11) \rightleftharpoons x(26)$
14	0 1 1 | 1 0	0 1 1 | 1 0	14	not needed
15	0 1 1 | 1 1	1 1 1 | 1 0	30	$x(15) \rightleftharpoons x(30)$

[a]Vertical dotted line denotes position of pivot.

address $\hat{k}$ of k is obtained immediately if one has a table for the bit reversal of an array of size $\sqrt{N} = 2^{M/2}$.

For example, Table IA) could be used to unscramble an array X of size $N = 256$. In this case $M = 8, M_1 = 4, N_1 = 16$. Assume that $k = 52$, then

$$k_2 = [k/16] = 3 \text{ and } k_1 = k - 16\,k_2 = 4.$$

From Table IA), $\hat{k}_1 = 2$ and $\hat{k}_2 = 12$. It follows that

$$\hat{k} = 2(16) + 12 = 44.$$

M Odd

When M is odd, the M bits are divided into the following three groups: a 1-b group on the pivot, and two groups of magnitude $M_1 = (M - 1)/2$ symmetrically placed about the pivot. An M-bit number k may be expressed as a function of a 1-b number j and two numbers k_1 and k_2 by

$$k = k_2 2N_1 + jN_1 + k_1 \tag{10}$$

where $k = 0, 1, \cdots, (N-1)$, $N_1 = 2^{M_1} = \sqrt{N/2}$, and $k_1, k_2 = 0, 1, \cdots, (N_1 - 1)$ and $j = 0, 1$. Again, after bit reversal and exchange of the least and most significant groups of bits, one obtains

$$k = k_2 2N_1 + jN_1 + k_1$$
$$\hat{k} = \hat{k}_1 2N_1 + jN_1 + \hat{k}_2 . \tag{11}$$

That is, if k is an address for an array of size $N = 2^M$, where M is odd, the bit reversed address $\hat{k}$ of k is obtained immediately if one has a table for bit reversal of an array of size $\sqrt{N/2} = 2^{(M-1)/2}$.

For example, Table IA) could be used to unscramble an array X of size $N = 512$. In this case $M = 9, M_1 = 4, N_1 = 16$. Assume that $k = 153$, then $k_2 = [k/32] = 4$, $j = [(k - k_2(32))/16] = 1$, and $k_1 = k - (k_2 32 + j16) = 9$. From Table IA), $\hat{k}_1 = 9, \hat{k}_2 = 2$. It follows that when $k = 153$,

$$\hat{k} = 9(32) + 16 + 2 = 306.$$

The formulas for one-step unscrambling are implemented efficiently in Section V.

IV. A General Formula for Unscrambling in Steps

The unscrambling of X defined by (6) can be performed in steps without using any auxiliary storage. Instead of a direct motion, the element $x(k)$ will be moved to location $\hat{k}$, when $\hat{k} > k$, via a series of intermediate locations. The locations for the intermediate exchanges are defined as k_1 and k_2 where k_2 differs from k_1 only in the reversal of groups of bits symmetric about the pivot. In each step the exchanges will be performed only for $k_2 > k_1$ so as to satisfy the condition that $\hat{k}$ be greater than k. In order to compute k_2 from k_1, it is necessary to express k_1 as a function of symmetric groups of bits, each group consisting of one or more bits.

Let the M bits of an address be divided into five groups of bits w_1, w_2, w_3, w_4, and w_5 of size M_1, M_2, M_3, M_2, and M_1, respectively, such that $M = 2M_1 + 2M_2 + M_3$, group w_3 straddling the pivot, w_1 and w_5, w_2, and w_4 being symmetrically placed about the pivot. If $N_i = 2^{M_i} (i = 1, 2, 3)$, then a general address k_1 can be written

$$k_1 = w_5 N_1 N_2{}^2 N_3 + w_4 N_1 N_2 N_3 + w_3 N_1 N_2 + w_2 N_1 + w_1 \tag{12}$$

where $0 \leqslant w_1, w_5 < N_1; 0 \leqslant w_2, w_4 < N_2; 0 \leqslant w_3 < N_3$.

Unscrambling in steps will be illustrated by the particular case where only the symmetric groups of bits w_2 and w_4 are subject to bit reversal, the other groups remaining unchanged. If w_4 is replaced by $\hat{w}_2$ and w_2 by $\hat{w}_4$ in (12), this partial bit reversal with respect to w_2 and w_4 changes k_1 into k_2. An exchange of elements is necessary only when $k_2 > k_1$, which implies $\hat{w}_2 > w_4$. Such an exchange is defined by

$$k_1 = w_5 N_1 N_2{}^2 N_3 + w_4 N_1 N_2 N_3 + w_3 N_1 N_2 + w_2 N_1 + w_1$$
$$k_2 = w_5 N_1 N_2{}^2 N_3 + \hat{w}_2 N_1 N_2 N_3 + w_3 N_1 N_2 + \hat{w}_4 N_1 + w_1 \tag{13}$$

where $w_1 = 0, 1, \cdots, (N_1 - 1); w_3 = 0, 1, \cdots, (N_3 - 1); w_5 =$

0, 1, ···, $(N_1 - 1)$; $\hat{w}_2 = (w_4 + 1), \cdots, (N_2 - 1)$; and $w_4 = 0, 1, \cdots, (N_2 - 2)$.

Assume that the unscrambling is performed in I steps, $i = 1, 2, \cdots, I$, starting with the least and most significant groups of bits. During the ith step, the bit reversal involves two symmetric groups of bits of size $M_2(i)$, such that

$$\sum_{i=1}^{I} M_2(i) = \left[\frac{M}{2}\right]$$

and the sizes of the five groups are defined by

$$M_1(1) = 0$$
$$M_1(i) = M_1(i-1) + M_2(i-1)$$
$$M_3(i) = M - 2M_1(i) - 2M_2(i).$$

Clearly, a particular set of values of w_1, w_3; and w_5 defines a block of elements, and w_2 and w_4 define the various elements inside each block. Notice, however, that the increment ($INC = k_2 - k_1$) does not depend on w_1, w_3 or w_5 and hence all the elements with the same specified w_2 and w_4, i.e., one pair of elements in each block, move by the same amount. Thus, it is more efficient to unscramble all the blocks at the same time, by exchanging the $N_1{}^2 N_3$ pairs of elements which correspond to the same value of INC. Equation (13) has been used to develop various algorithms to unscramble in steps by selecting different sets for $M_2(i)$.

For a simple illustration of (13), consider a scrambled array X of size $16 = 2^4$. Assume that the unscrambling will be performed in two steps, one pair of bits at a time, i.e., $I = 2$ and $M_2(1) = M_2(2) = 1$. The binary expansion of an address is given by

$$k = w_4 8 + w_3 4 + w_2 2 + w_1$$

where $w_i = 0, 1$. The first step of the unscrambling corresponds to the exchange of w_1 and w_4, when $\hat{w}_1 = w_1 = 1$ and $w_4 = 0$. It is performed by exchanging four pairs of elements at addresses defined by $k_1 = \nu + 1$ and $k_2 = 8 + \nu$, where $\nu = w_3 4 + w_2 2 = 0, 2, 4, 6$. The second step corresponds to the exchange of w_2 and w_3. It is performed by exchanging the pair of elements defined by $k_1 = w_4 8 + w_3 4 + w_2 2 + w_1$, $k_2 = w_4 8 + \hat{w}_2 4 + w_3 2 + w_1$ for $\hat{w}_2 = w_2 = 1$ and $w_3 = 0$, that is, $k_1 = w_4 8 + 2 + w_1$ and $k_2 = w_4 8 + 4 + w_1$. Table IIA) gives the values of k_1 and k_2 and Table IIB) shows how the indexes of the elements that are in scrambled order become ordered after two sets of exchanges. Unscrambling one pair of bits at a time is denoted as binary unscrambling and is implemented in Section V.

V. Fortran Programs for Unscrambling

Various unscrambling programs have been developed using (9), (11), and (13). Three of these programs are briefly discussed. UNSONE accomplishes the unscrambling in one step with a minimum number of exchanges. UNSBIN performs the unscrambling one pair of bits at a time, with a minimum amount of coding. UNSDRM is a modification of UNSBIN for use when the core is not large enough to hold the data to transform and it must be stored in peripheral storage.

Since the index 0 is not allowed in the computer, the addresses of exchanging elements are incremented by 1. A hybrid notation, mathematical and Fortran, is used to best unify the mathematical development and the corresponding Fortran implementation. The Fortran naming follows the mathematical notation.

Subroutine UNSONE (X,M)

This subroutine is based on Section III. It unscrambles a complex array X of size $N = 2^M$ in one step. The program works in the following two steps: 1) a bit-reversed array is constructed (or read), and 2) the array X is unscrambled. The listing is in Appendix I. By definition,

$$M1 = [M/2] \text{ and } N1 = 2{**}M1.$$

The program includes two unscrambling algorithms, the first which is based on (9) is for M even and the second which is based on (11) is for M odd. The algorithm for M even is written in cards 19 to 26. All the elements of X for which $k_1 > \hat{k}_2$ must be exchanged. Let $KU = \hat{k}_2 + 1$ and $KV = k_1 + 1$, then the ranges for KU and KV are $KU = 1, 2, \cdots, (N-1)$ and $KV = (KU + 1) \cdots N$. Define a bit-reversed array IX such as $IX(l+1) = \hat{l}N1$; then $k_2 N_1 = IX(\hat{k}_2 + 1)$ and $\hat{k}_1 N_1 = IX(k_1 + 1)$. Equation (9) can be written

$$KP1 = IX(KU) + KV$$
$$KP2 = IX(KV) + KU \tag{14}$$

which shows that $KP1$ and $KP2$ can be obtained easily for M even, given IX.

The algorithm for M odd is derived in a very similar manner. It is written in cards 28 to 40. In this case IX is defined as $IX(l+1) = 2\hat{l}N1$. For $j = 0$, (11) yields

$$KP1 = IX(KU) + KV$$
$$KP2 = IX(KV) + KU. \tag{15}$$

However, another pair of exchanges is defined by $j = 1$ which corresponds to an increment $N1$ of $KP1$ and $KP2$

$$KP1 = KP1 + N1$$
$$KP2 = KP2 + N1. \tag{16}$$

Either one of the two unscrambling algorithms requires the construction of a scaled bit-reversed array $IX(l+1) = \hat{l}NX$, where $NX = 2^{MX}$ and $MX = [(M+1)/2]$. The algorithm proposed is very compact. It is not the most efficient, but this is secondary because the dimension of IX is relatively small. A scaled bit-reversed array of size $2N_i$ can be obtained easily from a scaled bit reversed array of size N_i. More precisely, consider an array $IX(k_i + 1)$ of size N_i, where $k_i = 0, 1, \cdots, (N_i - 1)$, and an array $IX(k_{i+1} + 1)$ of size $N_{i+1} = 2N_i$, where k_{i+1}

TABLE II
BINARY UNSCRAMBLING

A) List of (k_1, k_2)					
Step 1	k_1	1	3	5	7
	k_2	8	10	12	14
Step 2	k_1	2	3	10	11
	k_2	4	5	12	13

B) Ordering of Array X by Exchange $x(k_1) \rightleftharpoons x(k_2)$																
Scrambled array	0	8	4	12	2	10	6	14	1	9	5	13	3	11	7	15
After one step	0	1	4	5	2	3	6	7	8	9	12	13	10	11	14	15
After two steps	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$= 0, 1, \cdots, (N_{i+1} - 1)$. If k_{i+1} is expressed as

$$k_{i+1} = jN_i + k_i \tag{17a}$$

then one can show that

$$\hat{k}_{i+1} = 2\hat{k}_i + j \tag{17b}$$

where $j = 0, 1$. Multiplication of (17b) by NX and use of IX yields

$$IX(k_i + 1) = IX(k_i + 1) + IX(k_i + 1)$$
$$IX(k_i + 1 + N_i) = IX(k_i + 1) + NX. \tag{18}$$

The corresponding algorithm is in cards 10 to 16. The size of the additional array IX is relatively small. For example, if $N = 2^{15} = 32\,768$, then the size of IX is $2^7 = 128$.

Subroutine UNSBIN(X,M)

This subroutine, listed in Appendix II, performs bit-reversal unscrambling of a complex array X of size $N = 2^M$ in steps of one pair of bits each. Again $X(KP1)$ and $X(KP2)$ are exchanged for $KP2 > KP1$ where $KP1 = k_1 + 1$, $KP2 = k_2 + 1$ and k_1 and k_2 are given by (13). Table II illustrates binary unscrambling for $N = 2^4 = 16$. Referring to (13), $M2 = 1$; $M1 = 0, 1, 2, \cdots, ([M/2] - 1)$ and $M3 = M - 2 - 2M1$.

In UNSBIN, $M1P = M1 + 1, N1 = 2^{M1}, N3 = 2^{M3}, N1T = N1 + N1, N1P = N1 + 1, N1M = N1 - 1, NS = 2^{(M-M1)}$, $NSH = NS/2$, and $INC = KP2 - KP1 = NSH - N1$. The unscrambling is performed in $M1PMX = [M/2]$ steps. Note that the same increment is used for the entire step, i.e., for $(N/4)$ pairs of exchanges. For each step, exchanges take place for $w_2 = 1$ and $w_4 = 0$ and are defined by

$$INC = NPH - N1$$
$$L1 = N1P + w_5 NS, \quad \text{for } w_5 = 0, 1, \cdots, N1M$$
$$L2 = L1 + w_3\, N1T, \quad \text{for } w_3 = 0, 1, \cdots, (N3 - 1)$$
$$KP1 = L2 + w_1, \quad \text{for } w_1 = 0, 1, \cdots, N1M$$
$$KP2 = KP1 + INC.$$

An interesting remark is that in each step NG groups of size NS are unscrambled identically and that the exchanges involve pack of elements of size NP, where $NG = NP = 2^{M1}$.

Subroutine UNSDRM

Subroutine UNSDRM, listed in Appendix III, is an adaptation of UNSBIN for unscrambling large arrays stored on a drum (or a disk), with a minimum number of drum-to-core transfers. In this program the array XDR, which must be unscrambled, is stored in the drum file $IDRUM$. The array X of dimension $NCOR = 2^{MCOR}$ represents the available core storage, $M > MCOR$. The subroutine $DRUM(KDR, NB, X(KX), IDRUM, IWR)$ is used to write on or read from the drum, by selecting the last argument as 1 or 2. The first two arguments define the location on the drum KDR and the number of elements to be written or read, NB. Then, NB elements are transferred from X to XDR or XDR to X in such a manner that $X(KX)$ corresponds to $XDR(KDR)$. The unscrambling is performed in $[M/2]$ steps. Let $MCORH = NCOR/2$ and let $IB1$ be the distance between the pair of elements to be exchanged. To minimize the number of transfers, one must distinguish between far pairs for which $IB1 > NCORH$ and near pairs for which $IB1 \leqslant NCORH$.

The unscrambling of the array $XDRM$ with respect to far pairs is performed one step at a time, group after group, $NCORH$ pairs at a time. That is, $NCORH$ elements of the first half of a group in $XDRM$ are read into the first half of X, $NCORH$ elements of the second half of the group at distance $IB1$ are read into the second half of X. Then, the even elements of the first half of X and the odd elements of the second half of X are exchanged. Finally, the two halves of X are written on the drum.

The unscrambling of the array $XDRM$ with respect to the near pairs requires only $T = N/NCOR$ round-trip transfers. When $IB1$ becomes equal to $NCORH$, $NCOR$ points are transferred

TABLE III

OPERATIONS/STORAGE FOR UNSCRAMBLING $N = 2^M, M' = \left[\frac{M}{2}\right]$

Subprograms / Operations or Storage	Reference [1]	Reference [2]	UNSBIN	UNSONE
Multiplication	$M-1$	$4N[(M-2)2^{M-1}+1]$		
Addition/Subtraction	even: $2M+\frac{5}{2}N-\frac{1}{2}\sqrt{N}$ odd: $2M+\frac{5}{2}N-\frac{1}{2}\sqrt{2N}$	$4N[(M-2)2^{M-1}+1]$	$\frac{N}{4}(M'+2)$[a]	even: $N+3\sqrt{N}$ odd: $N+\sqrt{2N}$
Transfers	even: $\frac{3}{2}(N-\sqrt{N})$ odd: $\frac{3}{2}(N-\sqrt{2N})$	even: $\frac{3}{2}(N-\sqrt{N})$ odd: $\frac{3}{2}(N-\sqrt{2N})$	$\frac{3}{4}M'N$	even: $\frac{3}{2}(N-\sqrt{N})$ odd: $\frac{3}{2}(N-\sqrt{2N})$
Table Look-Ups	N			even: $N-\sqrt{N}$ odd: $N-\sqrt{2N}$
Tests	N			
Auxiliary Storage	$3M$	N		$2^{M'}$

[a] Approximation.

to the core, and the unscrambling is completed on these *NCOR* elements. Then, the next *NCOR* elements are unscrambled. Repetition of this procedure T times completes the unscrambling.

VI. Comparison of Four Unscrambling Subprograms

The subprograms UNSONE and UNSBIN and two subprograms described in [1] and [2] are compared in a number of operations, in storage required, and in computer time.

In Table III are shown the number of operations and storage required to unscramble an array X of size N. Note that the number of transfers is three times the number of exchanges. Insignificant operations (such as required for the initialization of the subprograms), are not included in this table.

Inspection of Table III leads to the following conclusions. The subprogram [2] requires the largest number of operations and storage of an auxiliary array of size N. UNSBIN and the subprogram of [1] have comparable performance, the first being faster for small arrays, the latter being faster for large arrays. UNSONE has the least number of operations (no tests, a minimum of exchanges) and requires little auxiliary storage.

These conclusions have been confirmed by runs on the Univac-1108 Executive 8 System. In summarizing the characteristics of the four unscrambling subprograms, the following may be noted.

1) Over the range of array sizes studied, the running time of UNSONE does not exceed 35 percent of the running time of its nearest competitor (UNSBIN or subprogram of [1]). UNSONE can be easily adapted to unscrambling other than radix-2.

2) UNSBIN is markedly the most compact (25 cards) and can be used with peripheral storage devices to unscramble arrays too large for the core.

3) The subprogram of [1] follows most directly from the bit-reversal formula, is easily coded into Fortran, and possibly exemplifies the approach most often used.

4) The subprogram of [2] is more general than the other three in that it is not restricted to arrays of dimension $N = 2^M$, but suffers from the disadvantage of low running speed and the necessity of utilizing extra storage of magnitude N.

VII. Conclusions

Bit reversal of an array X of dimension N can be performed by exchanging the elements $x(k)$ and $x(\hat{k})$ for $\hat{k} > k$, where $\hat{k}$ is the address obtained by bit reversal of k. Bit reversal of an address is interpreted as an exchange of groups of bits symmetric with respect to a pivot. A formula based on the simultaneous exchange of all the symmetric bits is developed to compute $\hat{k}$ from k. Implementation of this formula results in an efficient program for the one-step bit reversal of X. A more general formula is presented for the exchange of two symmetric groups of bits. It is useful for unscrambling in steps where $x(k)$ is moved to location $\hat{k}$, via a series of intermediate locations. As an illustration, when the general formula is used for binary unscrambling, where bit reversal is performed in steps, then each step corresponds to the symmetric exchange of two bits. Binary unscrambling is ideal for unscrambling large arrays contained in external storage, because it minimizes the number of transfers between core and peripheral device. Computer programs for one-step and binary unscrambling are presented and compared to others described in the literature. The one-step unscrambling program requires fewer operations and is markedly faster than any other known to the authors. Fortran listings are included for reference.

APPENDIX I

```
      SUBROUTINE UNSONE(X,M)
      COMPLEX  X(1),XS
C DIMENSION 128 FOR IX ALLOWS TO UNSCRAMBLE ARRAYS OF SIZE UP TO 32768
      DIMENSION  IX(128)
      IF(M.EQ.1) RETURN
      M1=M/2
      MX=(M+1)/2
      N1=2**M1
      NX=2**MX
      IX(1)=0
      NI=1
      DO 3 I=1,M1
      DO 2 J=1,NI
      IX(J)=IX(J)+IX(J)
    2 IX(J+NI)=IX(J)+NX
    3 NI=NI+NI
      KUMX=N1-1
      IF (M1.NE.MX) GO TO 10
      DO 11 KU=1,KUMX
      KVMN=KU+1
      DO 11 KV=KVMN,N1
      KP1=IX(KU)+KV
      KP2=IX(KV)+KU
      XS=X(KP1)
      X(KP1)=X(KP2)
   11 X(KP2)=XS
      RETURN
   10 DO 12 KU=1,KUMX
      KVMN=KU+1
      DO 12 KV=KVMN,N1
      KP1=IX(KU)+KV
      KP2=IX(KV)+KU
      XS=X(KP1)
      X(KP1)=X(KP2)
      X(KP2)=XS
      KP1=KP1+N1
      KP2=KP2+N1
      XS=X(KP1)
      X(KP1)=X(KP2)
   12 X(KP2)=XS
      RETURN
      END
```

APPENDIX II

```
      SUBROUTINE UNSBIN(X,M)
      COMPLEX X(1),XS
      N=2**M
      M1PMX=M/2
      N1T=1
      NSH=N
      DO  1    M1P=1,M1PMX
      N1=N1T
      N1P=N1+1
      N1M=N1-1
      N1T=N1T+N1T
      NS=NSH
      NSH=NSH/2
      INC=NSH-N1
      DO  1    L1=N1P,N,NS
      L2MX=L1+INC
      DO  1    L2=L1,L2MX,N1T
      KP1MX=L2+N1M
      DO  1    KP1=L2,KP1MX
      KP2=KP1+INC
      XS=X(KP1)
      X(KP1)=X(KP2)
    1 X(KP2)=XS
      RETURN
      END
```

APPENDIX III

```
*     SUBROUTINE UNSORM(X,M,MCOR,IORUM)
*     COMPLEX X(1),XS
*     N=2**M
*     M2=M/2
*     NCOR=2**MCOR
*     NCORH=NCOR/2
*     NCORHP=NCORH+1
*     IA2=1
*     IB1=N
*     MD1F=M-MCOR
```

```
      M1PMX=MDIF
      IF(MDIF.GT.M2) M1PMX=M2
C NESTED DO LOOPS 1 CORRESPOND TO  IB2 .GT. NCOR  OR  IB1 .GT. NCORH
CARRAY XOR IN DRUM IS TRANSFORMED IN M1PMX STEPS OF ONE-PAIR OF BITS
      DO 1 M1P=1,M1PMX
      IA1=IA2
      IA1P1=IA1+1
      IA1M1=IA1-1
      IA2=IA2+IA2
      IB2=IB1
      IB1=IB1/2
      IB1M1=IB1-1
      INC=NCORH-IA1
C NCORH .GE. IA1
C WHEN NCORH.EQ.IA1,INC.EQ.0,EXCHANG LOWER WITH UPPER HALF OF ARRAY X
      IF (INC.EQ.0) GO TO 6
C  KSBLOC .LE. K1DR .LT. KSBLOC+IB1
      DO 2 KSBLOC=1,N,IB2
      K1DRMX=KSBLOC+IB1M1
      DO 2 K1DR=KSBLOC,K1DRMX,NCORH
      K2DR=K1DR+IB1
      CALL DRUM (K1DR,NCORH,X(1),IDRUM,2)
      CALL DRUM (K2DR,NCORH,X(NCORHP),IDRUM,2)
      DO 3 L2=IA1P1,NCORH,IA2
      KP1MX=L2+IA1M1
      DO 3 KP1=L2,KP1MX
      KP2=KP1+INC
      XS=X(KP1)
      X(KP1)=X(KP2)
    3 X(KP2)=XS
      CALL DRUM (K1DR,NCORH,X(1),IDRUM,1)
    2 CALL DRUM (K2DR,NCORH,X(NCORHP),IDRUM,1)
      GO TO 1
    6 DO 7 KSBLOC=1,N,IB2
      L1DRMN=KSBLOC+IA1
      K1DRMX=KSBLOC+IB1M1
      DO 7 L1DR=L1DRMN,K1DRMX,NCOR
      L2DR=L1DR+IB1-IA1
      CALL DRUM (L1DR,NCORH,X(1),IDRUM,2)
      CALL DRUM (L2DR,NCORH,X(NCORHP),IDRUM,2)
      CALL DRUM (L2DR,NCORH,X(1),IDRUM,1)
    7 CALL DRUM (L1DR,NCORH,X(NCORHP),IDRUM,1)
    1 CONTINUE
      IF(M1PMX.EQ.M2) RETURN
C SETS OF NCOR POINTS ARE BROUGT IN X AND  EACH SET IS UNSCRAMBLED
      M1PMN=M1PMX+1
      IA2S=IA2
      IB1S=IB1
      DO 4 K1DR=1,N,NCOR
      IA2=IA2S
      IB1=IB1S
      CALL DRUM(K1DR,NCOR,X(1),IDRUM,2)
      DO 5 M1P=M1PMN,M2
      IA1=IA2
      IA1P1=IA1+1
      IA1M1=IA1-1
      IA2=IA2+IA2
      IB2=IB1
      IB1=IB1/2
      INC=IB1-IA1
      DO 5 L1=IA1P1,NCOR,IB2
      L2MX=L1+INC
      DO 5 L2=L1,L2MX,IA2
      KP1MX=L2+IA1M1
      DO 5 KP1=L2,KP1MX
      KP2=KP1+INC
      XS=X(KP1)
      X(KP1)=X(KP2)
    5 X(KP2)=XS
    4 CALL DRUM(K1DR,NCOR,X(1),IDRUM,1)
      RETURN
```

References

[1] W.M. Gentleman and G. Sande, "Fast Fourier transforms for fun and profit," in *1966 Fall Joint Computer Conf., AFIPS Conf. Proc.*, vol. 29. Washington, D.C.: Spartan, 1966, pp. 563-578.

[2] D.K. Kahaner, "Matrix description of the fast Fourier transform," *IEEE Trans. Audio Electroacoust.*, vol. AU-18, pp. 442-450, Dec. 1970.

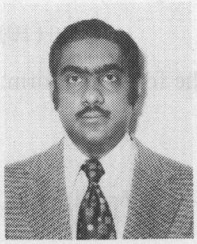
[3] J.W. Cooley and J.W. Tukey, "An algorithm for the machine calculation of complex Fourier seriers," *Math. Comput.*, vol. 90, pp. 297-301, 1965.

Robert J. Polge (M'64) was born in Anduze, France, on March 11, 1928. He received the Bachelor of Science degree and License es Sciences from the University of Montpellier, France, in 1946 and 1950, respectively; the diploma of Ingenieur E.S.E. from the Ecole Superieure d'Electricite de Paris, France, in 1952; and the M.S. and Ph.D. degrees from Carnegie-Mellon University, Pittsburgh, Pa., in 1961 and 1963, respectively.

In 1953 he joined the Societe Andre

Citroen, Paris, as a Research Engineer working with the control of machine tools. In 1956 he joined the Societe Sciaky, Paris, as Head of the Electronics Laboratory, to design and develop circuits for welding machines. He spent 1959 and 1960 with the Compagnie Generale de Telegraphie sans Fil (CSF), as Head of the Technology Department. From 1961 to 1963 he was a Teaching Assistant at Carnegie-Mellon University. In 1963 joined the University of Alabama, Huntsville, as an Assistant Professor of Electrical Engineering; he became Professor in 1967. He teaches and conducts research in the area of communications and data processing.

Dr. Polge is a member of the Societe Francaise des Electriciens and Sigma Xi.



B.K. Bhagavan was born in Mysore, India, on February 15, 1947. He received the B.E. degree in electrical engineering from the Bangalore University, Bangalore, India, in 1967, the M.E. degree in electrical power engineering from the Indian Institute of Science, Bangalore, in 1969, and the Ph.D. degree from Southern Methodist University, Dallas, Tex., in 1971.

Presently he is a Research Associate with the Research Insitute of the University of Alabama, Hunstsville. His current interests include optimal control, digital processing of image data, and simulation and analysis of radar systems.

James M. Carswell was born in Bishop, Calif., on January 19, 1922.

His major field of interest is mechanical engineering, although he has had considerable experience in electrical engineering. For the last 15 years his work has included trajectory analysis; in particular, digital simulation of six-degree-of freedom trajectories, Monte Carlo analysis of trajectory variables, and special problems in flight mechanics. Until June 1973 he was with the University of Alabama, Huntsville. Currently he is with North American Aviation, Downey, Calif.

Floating-Point Arithmetic Algorithms in the Symmetric Residue Number System

EISUKE KINOSHITA, HIDEO KOSAKO, MEMBER, IEEE, AND YOSHIAKI KOJIMA, SENIOR MEMBER, IEEE

Abstract—**The residue number system is an integer number system and is inconvenient to represent numbers with fractional parts. In the symmetric residue system, a new representation of floating-point numbers and arithmetic algorithms for its addition, subtraction, multiplication, and division are proposed. A floating-point number is expressed as an integer multiplied by a product of the moduli. The proposed system assumes existence of necessary conversion procedures before and after the computation.**

Index Terms—**Cyclic mixed-radix system, exponent part, floating-point arithmetic algorithms, floating-point representation, mantissa, normalized form, number of precision n, symmetric residue number system.**

Manuscript received September 10, 1971; revised August 4, 1973.

The authors are with the Department of Electronics, University of Osaka Prefecture, Osaka, Japan.

I. Introduction

THE residue number system is an integer number system. At present, the techniques known make it inconvenient to represent fractional quantities. It is to be desired that numbers with fractional parts can be handled as easily as integers in the residue number systems.

A few studies on the floating-point arithmetic in the residue system have been published [1], [2]. In these reports a power of 2 or 10 is used as an exponent.

This paper deals with floating-point arithmetic with an exponent which is a product of moduli in the symmetric residue number system. This number system has the following advantages: 1) finding the additive inverse of a residue digit is fairly easy, 2) sign detection by mixed-radix conversion is