RICE UNIVERSITY

A PRIME FACTOR FFT ALGORITHM USING HIGH SPEED CONVOLUTION

by

Dean P. Kolba

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

Master of Science

Thesis Director's Signature

Houston, Texas

May, 1977

ABSTRACT

A PRIME FACTOR FFT ALGORITHM
USING HIGH SPEED CONVOLUTION

by Dean P. Kolba

Two recently developed ideas; the conversion of a DFT to convolution and the implementation of short convolutions with a minimum of multiplications, are combined to give efficient algorithms for long transforms. Three transform algorithms are compared in terms of number of multiplications and additions. Timing for a prime factor FFT algorithm using high speed convolution, which was programmed for an IBM 370 and an 8080 microprocessor,is presented.

ACKNOWLEDGEMENTS

I would like to thank my research advisor, Dr. T.W. Parks, who provided invaluable guidance and assistance in the completion of this research, and my fiancée, Jan DeMoss, who has supported me during the preparation of this thesis.

I. INTRODUCTION

The calculation of the Discrete Fourier Transform (DFT)

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j\frac{2\pi}{N}nk}$$

is one of the central operations in digital signal processing. The development and widespread use of the Fast Fourier Transform, stimulated by the paper of Cooley and Tukey [1], has had a major impact on signal processing.

Recently, several new ideas have emerged which lead to new algorithms for the DFT. One key idea described by Rader [2] in 1968 was the observation that computation of the DFT can be changed into a circular convolution by rearranging the data when N is prime. Thus, if one has a fast way to do convolution, he now has a fast way to do the DFT.

Winograd [3] has shown the minimum number of multiplications required for circular convolution. New convolution algorithms which often achieve this minimum are being developed by Agarwal and Cooley [4] .

In a concise paper, Winograd [5] combines the conversion of a DFT to convolution, for prime and prime power lengths, with these new convolution algorithms for short transforms. He proposes that long transforms be computed by nesting these short, high speed transforms and presents a table comparing the number of operations required with the conventional Fast Fourier Transform (FFT).

This thesis first reviews the two central ideas; conversion of a DFT to circular convolution and convolution with the minimum number of multiplications, then presents a study of various implementations of long transforms.

An alternative to the nested algorithm proposed by Winograd, a prime factor FFT algorithm using high speed convolution for individual factors, is singled out as a promising approach and programmed for two machines; an IBM 370 and an 8080 microprocessor. Tables compare this approach with Winograd's nesting and with the conventional power of 2 FFT. While the idea of breaking up a one dimensional transform into a multidimensional transform with prime factors is not new- see Good [6] and Thomas [7], the combination of short, high speed convolution algorithms with this multidimensional expansion appears to be a promising new way to implement the DFT.

II. DOING DFT'S WITH CONVOLUTION

A. Prime Length DFT

The Discrete Fourier Transform

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W^{nk} \qquad k=0,1,\cdots,N-1 \tag{1}$$

where $W = e^{-\frac{j2\pi}{N}}$

is a linear transformation of the N-dimensional data vector

$\underline{x} = \begin{bmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{bmatrix}$ into the vector $\underline{X} = \begin{bmatrix} X(0) \\ X(1) \\ \vdots \\ X(N-1) \end{bmatrix}$ of frequency samples.

The matrix representation of the DFT

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W^1 & W^2 & \cdots & W^{N-1} \\ 1 & W^2 & W^4 & \cdots & W^{2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & W^{N-1} & W^{2(N-1)} & \cdots & W^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{bmatrix} \tag{2}$$

shows the complexity of the computation arises from the (N-1) by (N-1) lower right portion of the matrix. If

$$\overline{X}(k) = \sum_{n=1}^{N-1} x(n)\, W^{nk} \qquad k=1,2,\cdots,N-1 \tag{3}$$

can be computed efficiently, then we will have a fast algorithm for computing (1), the DFT, since from (3), (1) can easily be calculated:

$$X(0) = \sum_{n=0}^{N-1} x(n) \tag{4}$$

$$X(k) = x(0) + \overline{X}(k) \qquad k=1,2,\cdots,N-1$$

To see how (3) may be converted to circular convolution, consider the matrix representation of (3), for N=5 with exponents taken modulo 5:

$$\begin{bmatrix} \overline{X}(1) \\ \overline{X}(2) \\ \overline{X}(3) \\ \overline{X}(4) \end{bmatrix} = \begin{bmatrix} W^1 & W^2 & W^3 & W^4 \\ W^2 & W^4 & W^1 & W^3 \\ W^3 & W^1 & W^4 & W^2 \\ W^4 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ x(3) \\ x(4) \end{bmatrix} \tag{5}$$

If we interchange the last two columns of (5) and then interchange the last two rows, we get

$$\begin{bmatrix} \overline{X}(1) \\ \overline{X}(2) \\ \overline{X}(4) \\ \overline{X}(3) \end{bmatrix} = \begin{bmatrix} W^1 & W^2 & W^4 & W^3 \\ W^2 & W^4 & W^3 & W^1 \\ W^4 & W^3 & W^1 & W^2 \\ W^3 & W^1 & W^2 & W^4 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ x(4) \\ x(3) \end{bmatrix} \tag{6}$$

We now have something like backwards circular convolution, or circular correlation. By fixing x(1) and reversing the remaining input vector we obtain the conventional circular convolution:

$$\begin{bmatrix} \bar{X}(1) \\ \bar{X}(2) \\ \bar{X}(4) \\ \bar{X}(3) \end{bmatrix} = \begin{bmatrix} W^1 & W^3 & W^4 & W^2 \\ W^2 & W^1 & W^3 & W^4 \\ W^4 & W^2 & W^1 & W^3 \\ W^3 & W^4 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(3) \\ x(4) \\ x(2) \end{bmatrix} \qquad (7)$$

The computation of (3) (and thus the DFT) has been changed to a circular convolution (7) by permuting the input and the output indices. If we have a fast way of doing convolutions we now have a fast way of doing DFT's.

This idea was first presented by Rader [2]. He showed how the permutation can always be done when N is a prime number. Since $W^N = 1$ we are dealing with the product of integers modulo N in the exponent of W in (3). To change the DFT into a circular convolution a mapping of the indices is used to change multiplication of indices modulo N to addition of indices modulo N-1. The set of integers $\{1,2,\cdots,N-1\}$ forms a cyclic group with the operation of multiplication modulo N [8]. We can always find at least one integer α in the group with the property that any integer in the group may be expressed as some power of α. By ordering the data according to the exponent of α we can always change (3) to circular convolution for prime N. The relationship between the new index m and the original n is

$$n = \alpha^m \text{ modulo } N \qquad \begin{matrix} n=1,2,\cdots,N-1 \\ m=0,1,\cdots,N-2 \end{matrix} \qquad (8)$$

where $$\begin{matrix} \alpha^k \neq 1 \quad \text{for } 0 < k < N-1 \\ \alpha^{N-1} = 1 \end{matrix} \qquad (9)$$

The permutation map (8) is an isomorphism between the multiplicative group of N-1 integers $\{1,2,\cdots,N-1\}$ and the additive group of N-1 integers $\{0,1,\cdots,N-2\}$. An integer α with the property (9) is called a primitive $(N-1)^{st}$ root of unity and is said to generate the group since any element can be written as some power of α. Just as logarithms change multiplication to addition, (8) changes multiplication into addition of indices. When (8) is used on both input and output indices, (3) becomes

$$\bar{X}(\alpha^{l}) = \sum_{m=0}^{N-2} x(\alpha^{m})\, W^{\alpha^{(l+m)}} \qquad l=0,1,\cdots,N-2 \qquad (10)$$

In (10) the exponents of α are taken modulo N-1. This gives, for N=5, the backward circular convolution of (6) when $\alpha = 2$. To obtain conventional circular convolution we change the sign of m in (10) which corresponds to fixing $x(\alpha^{0})$ and reversing the remaining input sequence.

$$\bar{X}(\alpha^{l}) = \sum_{m=0}^{N-2} x(\alpha^{-m})\, W^{\alpha^{(l-m)}} \qquad l=0,1,\cdots,N-2 \qquad (11)$$

Again, the indices in (11), (the exponents of α) are taken modulo N-1. By combining (11) with (4) we can always convert the computation of a DFT of prime length into a circular convolution.

B. Prime Power Length DFT

Winograd [5] and Rader and McClellan [9] have shown that the DFT may also be converted to convolution when the transform length N is a prime power, i.e. $N = p^r$ for a prime $p \neq 2$. The conversion is a bit more complicated since we must first remove all integers which contain a factor p from the set $\{1,2,\cdots,N-1\}$ to get a cyclic group with $p^{r-1}(p-1)$ elements. This cyclic group leads to a circular convolution of length $p^{r-1}(p-1)$ as before. The remaining computation consists of two DFT's of length p^{r-1}. For example with $N = 9 = 3^2$, we delete the integers 3 and 6 to obtain the set $\{1,2,4,5,7,8\}$ which forms a cyclic group under multiplication modulo 9 and is isomorphic to the additive group of integers $\{0,1,2,3,4,5\}$ under addition modulo 6. The integer 2 will generate the multiplicative group since the powers of 2 modulo 9, $2^m \bmod 9$, $m = 0,1,\cdots,5$, are 1,2,4,8,7,5 .

In terms of the matrix representation

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \\ X(8) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 & W^8 \\ 1 & W^2 & W^4 & W^6 & W^8 & W^1 & W^3 & W^5 & W^7 \\ 1 & W^3 & W^6 & 1 & W^3 & W^6 & 1 & W^3 & W^6 \\ 1 & W^4 & W^8 & W^3 & W^7 & W^2 & W^6 & W^1 & W^5 \\ 1 & W^5 & W^1 & W^6 & W^2 & W^7 & W^3 & W^8 & W^4 \\ 1 & W^6 & W^3 & 1 & W^6 & W^3 & 1 & W^6 & W^3 \\ 1 & W^7 & W^5 & W^3 & W^1 & W^8 & W^6 & W^4 & W^2 \\ 1 & W^8 & W^7 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \\ x(8) \end{bmatrix} \quad (12)$$

we remove rows and columns corresponding to indices 0,3, and 6 and compute the remaining length 6 transformation

$$\begin{bmatrix} \overline{X}(1) \\ \overline{X}(2) \\ \overline{X}(4) \\ \overline{X}(5) \\ \overline{X}(7) \\ \overline{X}(8) \end{bmatrix} = \begin{bmatrix} W^1 & W^2 & W^4 & W^5 & W^7 & W^8 \\ W^2 & W^4 & W^8 & W^1 & W^5 & W^7 \\ W^4 & W^8 & W^7 & W^2 & W^1 & W^5 \\ W^5 & W^1 & W^2 & W^7 & W^8 & W^4 \\ W^7 & W^5 & W^1 & W^8 & W^4 & W^2 \\ W^8 & W^7 & W^5 & W^4 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ x(4) \\ x(5) \\ x(7) \\ x(8) \end{bmatrix} \tag{13}$$

using the permutation

$$n = 2^m \bmod 9 \quad m=0,1,2,3,4,5 \quad ; \; n=1,2,4,8,7,5$$

to obtain the circular convolution (with input reversed as before)

$$\begin{bmatrix} \overline{X}(1) \\ \overline{X}(2) \\ \overline{X}(4) \\ \overline{X}(8) \\ \overline{X}(7) \\ \overline{X}(5) \end{bmatrix} = \begin{bmatrix} W^1 & W^5 & W^7 & W^8 & W^4 & W^2 \\ W^2 & W^1 & W^5 & W^7 & W^8 & W^4 \\ W^4 & W^2 & W^1 & W^5 & W^7 & W^8 \\ W^8 & W^4 & W^2 & W^1 & W^5 & W^7 \\ W^7 & W^8 & W^4 & W^2 & W^1 & W^5 \\ W^5 & W^7 & W^8 & W^4 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(5) \\ x(7) \\ x(8) \\ x(4) \\ x(2) \end{bmatrix} \tag{14}$$

In addition to (14) we must complete the computation for the rows and columns removed from (12). For the deleted rows we have

$$\begin{bmatrix} X(0) \\ X(3) \\ X(6) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W^3 & W^6 & 1 & W^3 & W^6 & 1 & W^3 & W^6 \\ 1 & W^6 & W^3 & 1 & W^6 & W^3 & 1 & W^6 & W^3 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \\ x(8) \end{bmatrix} \tag{15}$$

Since $W^3 = e^{\frac{-j2\pi}{9}3} = e^{\frac{-j2\pi}{3}} = W_3$, (15) is simply a length 3 DFT of added data

$$\begin{bmatrix} X(0) \\ X(3) \\ X(6) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & W_3 & W_3^2 \\ 1 & W_3^2 & W_3 \end{bmatrix} \begin{bmatrix} x(0)+x(3)+x(6) \\ x(1)+x(4)+x(7) \\ x(2)+x(5)+x(8) \end{bmatrix}$$

For the deleted columns we have

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \\ Y(4) \\ Y(5) \\ Y(6) \\ Y(7) \\ Y(8) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & W^3 & W^6 \\ 1 & W^6 & W^3 \\ 1 & 1 & 1 \\ 1 & W^3 & W^6 \\ 1 & W^6 & W^3 \\ 1 & 1 & 1 \\ 1 & W^3 & W^6 \\ 1 & W^6 & W^3 \end{bmatrix} \begin{bmatrix} x(0) \\ x(3) \\ x(6) \end{bmatrix} \tag{16}$$

For (16) a second length 3 DFT can be computed.

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \end{bmatrix} = \begin{bmatrix} Y(3) \\ Y(4) \\ Y(5) \end{bmatrix} = \begin{bmatrix} Y(6) \\ Y(7) \\ Y(8) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & W_3 & W_3^2 \\ 1 & W_3^2 & W_3 \end{bmatrix} \begin{bmatrix} x(0) \\ x(3) \\ x(6) \end{bmatrix} \tag{17}$$

Only the last two entries Y(1) and Y(2) are needed from (17) to complete (12) using (14)

$$\begin{bmatrix} X(1) \\ X(2) \\ X(4) \\ X(5) \\ X(7) \\ X(8) \end{bmatrix} = \begin{bmatrix} \bar{X}(1) \\ \bar{X}(2) \\ \bar{X}(4) \\ \bar{X}(5) \\ \bar{X}(7) \\ \bar{X}(8) \end{bmatrix} + \begin{bmatrix} Y(1) \\ Y(2) \\ Y(1) \\ Y(2) \\ Y(1) \\ Y(2) \end{bmatrix}$$

This length 9 example thus requires a length 6 circular convolution and 2 length 3 DFT's which can be computed using length 2 circular convolutions. If $N = 3^3$, we have a length $3^2 \cdot 2 = 18$ circular convolution and two length $3^2 = 9$ DFT's. These can be reduced to two length 6 convolutions and 4 length 3 DFT's which are calculated with length 2 circular convolutions. If $N = p^r$, the length N transform is computed with 1 length $p^{r-1}(p-1)$ convolution, 2 length $p^{r-2}(p-1)$ convolutions, 4 length $p^{r-3}(p-1)$ convolutions, 8 length $p^{r-4}(p-1)$ convolutions, $\cdots$, terminating with 2^{r-1} length p-1 convolutions.

III. CONVOLUTION WITH THE MINIMUM NUMBER OF MULTIPLICATIONS

The prime and prime power DFT algorithms are means of converting the calculations required in the DFT into circular convolution. Then, special fast circular convolution techniques may be used to preform the calculations. An algorithm for computing a short length circular convolution in the minimum number of multiplications for small values of N is based on recent work by Winograd [3].

Winograd's theorem on the minimum number of multiplications is explained in terms of polynomial multiplication. To cyclicly convolve the sequences $h_0, h_1, \cdots, h_{N-1}$ and $x_0, x_1, \cdots, x_{N-1}$ we need only find the N coefficients of the polynomial

$$Y(z) = H(z) \cdot X(z) \text{ modulo } (z^N - 1) \tag{18}$$

where

$$H(z) = \sum_{k=0}^{N-1} h_k z^k \quad \text{and} \quad X(z) = \sum_{k=0}^{N-1} x_k z^k \tag{19}$$

If $z^N - 1$ is written in terms of the K factors which are irreducible over the rationals

$$z^N - 1 = \prod_{i=1}^{K} Q_i(z) \tag{20}$$

with no common factors in the $Q_i(z)$ polynomials, Winograd's Theorem states that the minimum number of multiplications required to compute the circular convolution of two length N sequences is 2N-K. This theorem does not count multiplication by rational numbers.

In order to reduce the computation required for (18), Y(z) is decomposed into K simpler parts using the polynomial version of the Chinese Remainder Theorem [4].

$$Y(z) = \left[\sum_{i=1}^{K} Y_i(z)\, S_i(z)\right] \bmod (z^N-1) \tag{21}$$

$$Y_i(z) = H_i(z)\, X_i(z) \bmod Q_i(z) \qquad i = 1,2,\cdots,K \tag{22}$$

$$\left.\begin{aligned} X_i(z) &= X(z) \bmod Q_i(z) \qquad i = 1,2,\cdots,K \\ H_i(z) &= H(z) \bmod Q_i(z) \qquad i = 1,2,\cdots,K \end{aligned}\right. \tag{23}$$

The polynomials $S_i(z)$ $i = 1,2,\cdots,K$ play the role of a Kroneker delta

$$\left.\begin{aligned} S_i(z) &= 1 \quad \bmod Q_i(z) \quad i = 1,2,\cdots,K \\ S_i(z) &= 0 \quad \bmod Q_j(z) \quad \text{for all } j \neq i \end{aligned}\right. \tag{24}$$

The $S_i(z)$ may be found by applying Euclid's algorithm to polynomials [4].

As an example, we will do a length 6 circular convolution of the sequences $h_0, h_1, \cdots, h_5$ and $x_0, x_1, \cdots, x_5$. We have the polynomials

$$H(z) = h_0 + h_1 z + \cdots + h_5 z^5$$

$$X(z) = x_0 + x_1 z + \cdots + x_5 z^5$$

First, the irreducible factors $Q_i(z)$ are found:

$$\begin{aligned} z^6-1 &= (z+1)(z-1)(z^2+z+1)(z^2-z+1) \\ &= Q_1(z)Q_2(z)\ Q_3(z)\ Q_4(z) \end{aligned}$$

Next, the intermediary polynomials X_i, H_i, and Y_i are formed:*

$$\begin{aligned} X_1(z) &= X(z) \bmod (z+1) = x_0^1 \\ &= x_0 - x_1 + x_2 - x_3 + x_4 - x_5 \\ X_2(z) &= X(z) \bmod (z-1) = x_0^2 \\ &= x_0 + x_1 + x_2 + x_3 + x_4 + x_5 \\ X_3(z) &= X(z) \bmod (z^2+z+1) = x_0^3 + x_1^3 z \\ &= (x_0 - x_2 + x_3 - x_5) + (x_1 - x_2 + x_4 - x_5)z \\ X_4(z) &= X(z) \bmod (z^2-z+1) = x_0^4 + x_1^4 z \\ &= (x_0 - x_2 - x_3 + x_5) + (x_1 + x_2 - x_4 - x_5)\ z \end{aligned} \tag{25}$$

(The $H_i(z)$ polynomials are the same in form.)

$$\begin{aligned} Y_1(z) &= H_1(z)\ X_1(z) \bmod (z+1) = y_0^1 = h_0^1\ x_0^1 \\ Y_2(z) &= H_2(z)\ X_2(z) \bmod (z-1) = y_0^2 = h_0^2\ x_0^2 \\ Y_3(z) &= H_3(z)\ X_3(z) \bmod (z^2+z+1) = y_0^3 + y_1^3\ z \\ &= (h_0^3\ x_0^3 - h_1^3\ x_1^3) + (h_1^3\ x_0^3 + h_0^3\ x_1^3 - h_1^3\ x_1^3)\ z \\ Y_4(z) &= H_4(z)\ X_4(z) \bmod (z^2-z+1) = y_0^4 + y_1^4\ z \\ &= (h_0^4\ x_0^4 - h_1^4\ x_1^4) + (h_1^4\ x_0^4 + h_0^4\ x_1^4 + h_1^4\ x_1^4)\ z \end{aligned} \tag{26}$$

* Superscripts are used to identify the i^{th} polynomial.

Now, the $Y_1(z)$ and $Y_2(z)$ polynomials are formed directly with one multiplication each. So, we have the intermediate variables

$$m_1 = h_0^1 x_0^1 \quad \text{and} \quad m_2 = h_0^2 x_0^2$$

giving

$$y_0^1 = m_1 \quad \text{and} \quad y_0^2 = m_2 \tag{27}$$

The polynomials $Y_3(z)$ and $Y_4(z)$ require only three multiplications each - similar to complex multiplication done in three real multiplications. Thus, for $Y_3(z)$ we need

$$\begin{aligned} m_3 &= (h_0^3 - h_1^3)\,(x_1^3 - x_0^3) \\ m_4 &= h_0^3 x_0^3 \quad \text{and} \quad m_5 = h_1^3 x_1^3 \end{aligned} \tag{28}$$

For $Y_4(z)$ we need

$$\begin{aligned} m_6 &= (h_0^4 + h_1^4)\,(x_0^4 + x_1^4) \\ m_7 &= h_0^4 x_0^4 \quad \text{and} \quad m_8 = h_1^4 x_1^4 \end{aligned} \tag{29}$$

From these intermediate variables we obtain

$$\begin{aligned} y_1^3 &= m_3 + m_4 \quad \text{and} \quad y_0^3 = m_4 - m_5 \\ y_1^4 &= m_6 - m_7 \quad \text{and} \quad y_0^4 = m_7 - m_8 \end{aligned} \tag{30}$$

At this point we have the four component polynomials:

$$Y_1(z) = y_0^1 \qquad Y_2(z) = y_0^2$$

$$Y_3(z) = y_0^3 + y_1^3\, z \qquad Y_4(z) = y_0^4 + y_1^4\, z$$

The final step is to express the polynomial Y(z) in terms of the components $Y_i(z)$. We have

$$Y(z) = \left[Y_1(z)\ S_1(z) + Y_2(z)\ S_2(z) + Y_3(z)\ S_3(z) + Y_4(z)\ S_4(z)\right] \bmod (z^6-1) \tag{31}$$

The $S_i(z)$ polynomials satisfying (24) are

$$\begin{aligned} S_1(z) &= -1/6\ (z^5-z^4+z^3-z^2+z-1) \\ S_2(z) &= 1/6\ (z^5+z^4+z^3+z^2+z+1) \\ S_3(z) &= 1/6\ (z^6-z^5-z^4+2z^3-z^2-z+1) \\ S_4(z) &= 1/6\ (z^6+z^5-z^4-2z^3-z^2+z+1) \end{aligned} \tag{32}$$

In order to show the exact operations on the original $\{x_i\}$ and $\{h_i\}$ the vector of coefficients of Y(z), $\underline{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix}$ may be expressed in terms of $\underline{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}$ and $\underline{h} = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{N-1} \end{bmatrix}$ with two equations using $\otimes$ to indicate point by point multiplication of column vectors:

$$\underline{m} = \left[B\right] \underline{h} \otimes \left[A\right] \underline{x} \tag{33}$$

$$\underline{y} = \left[\ C\ \right] \underline{m} \tag{34}$$

A length N circular convolution requiring M multiplies can always be expressed this way with A and B MxN matrices and C an NxM matrix [4].

In order to put our example for N=6 in this matrix form, we first identify the intermediate m parameters from (26), as shown in (27), (28), and (29). The terms in (26) involving h and x are expanded according to (25) to obtain the matrices A and B.

$$\underline{m} = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 0 & 1 & -1 & 0 \\ 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & 1 & -1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 & -1 & 0 \\ 1 & 0 & -1 & -1 & 0 & 1 \\ 0 & 1 & 1 & 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & 0 & -1 & 1 & 0 \\ 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & 1 & -1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 & -1 & 0 \\ 1 & 0 & -1 & -1 & 0 & 1 \\ 0 & 1 & 1 & 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \qquad (35)$$

To obtain the C matrix rewrite (26) in terms of the m's given by (27), (28), and (29).

$$\begin{aligned} Y_1(z) &= m_1 \\ Y_2(z) &= m_2 \\ Y_3(z) &= (m_4 - m_5) + (m_3 + m_4)\, z \\ Y_4(z) &= (m_7 - m_8) + (m_6 - m_7)\, z \end{aligned} \qquad (36)$$

Substitute (36) and (32) into (31), collect powers of z to obtain

$$\underline{y} = C\ \underline{m}$$

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 1 & -1 & 1 & -2 & 1 & 1 & -2 \\ -1 & 1 & 2 & 1 & 1 & 2 & -1 & -1 \\ 1 & 1 & -1 & -2 & 1 & 1 & -2 & 1 \\ -1 & 1 & -1 & 1 & -2 & -1 & -1 & 2 \\ 1 & 1 & 2 & 1 & 1 & -2 & 1 & 1 \\ -1 & 1 & -1 & -2 & 1 & -1 & 2 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \end{bmatrix} \tag{37}$$

We now have a length six circular convolution computed with 8 multiplications, which is the minimum number as given by Winograd's theorem:

$$2N - K = 2(6) - 4 = 8 \quad .$$

The multiplications by the rational numbers in A, B, and C are not counted.

The calculation of $Y_3(z)$ and $Y_4(z)$ may be done with a number of different algorithms. These will give values for m_3 through m_8 different from (28) and (29) and different evaluations of $Y_3(z)$ and $Y_4(z)$ in terms of the m's. This freedom in the choice of calculating $Y_3(z)$ and $Y_4(z)$ may be used to try to minimize the number of additions for the convolution algorithm. No attempt has been made here to minimize the number of additions.

Convolution algorithms which achieve 2N-K multiplications and have simple A,B, and C matrices are known only for short lengths. For longer convolutions it is difficult to see how to keep the number of additions under control.

IV. APPLICATION TO THE DFT

The polynomial method of generating a circular convolution algorithm which requires the minimum number of multiplications has been used effectively for short lengths. Winograd [5] states that all known algorithms for computing circular convolution in the minimum number of multiplications require a large number of additions when z^N-1 has large irreducible factors. Therefore to be of practical interest, the DFT length will be kept small to take advantage of the change to a circular convolution. DFT algorithms for small values of N have been written with these methods for use in a prime factor FFT algorithm to be described in Chapter V. These algorithms are different from those used for the nested DFT proposed by Winograd and implemented by Silverman [10], which is also described in Chapter V. Table 1 shows the number of operations required for short transforms intended for use in the prime factor FFT algorithm and for short transforms intended for use in the nested algorithm. Explicit formulas for short transforms to be used with a prime factor FFT are given for lengths 3,5,7, and 9 in Appendix A. These formulas come from combining the correction terms with special convolution algorithms described in Chapter III. Short transforms for use with the nesting algorithm are derived by modifying the transforms in Appendix A to reduce the number of W^0 multiplications as in (45) - (47).

Table 1

Short Length DFT Operations Count

	Prime Factor FFT			Nested Algorithm		
N	Multiplies	Shifts	Adds	Multiplies	W^0 multiplies	Adds
2	0	0	2	0	2	2
3	1	1	6	2	1	6
4	0	0	8	0	4	8
5	4	2	17	5	1	17
7	8	0	36	8	1	36
8	2	0	26	2	6	26
9	8	2	49	10	2	49

To see how a DFT is implemented with a convolution algorithm, concider the following length 3 example:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 \\ W^0 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \end{bmatrix}$$

The convolution

$$\begin{bmatrix} \overline{X}(1) \\ \overline{X}(2) \end{bmatrix} = \begin{bmatrix} W^1 & W^2 \\ W^2 & W^1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \end{bmatrix} \tag{38}$$

provides $\overline{X}(1)$ and $\overline{X}(2)$ and

$$
\begin{aligned}
X(0) &= W^0(x(0) + x(1) + x(2)) \\
X(1) &= W^0\, x(0) + \overline{X}(1) \\
X(2) &= W^0\, x(0) + \overline{X}(2)
\end{aligned}
\tag{39}
$$

To get an explicit formula for the DFT, the convolution (38) is written in matrix form as in (33) and (34).

$$
\begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} W^1 \\ W^2 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \end{bmatrix}
\tag{40}
$$

$$
\begin{bmatrix} \overline{X}(1) \\ \overline{X}(2) \end{bmatrix} = \tfrac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \end{bmatrix}
\tag{41}
$$

In applying (40) and (41) to the length 3 DFT we absorb the factor of $\frac{1}{2}$ into the B matrix. Then, combining this convolution with (39) gives the following length 3 DFT algorithm for use in the prime factor FFT.

$$
\begin{aligned}
a_1 &= x(1) + x(2) \\
a_2 &= x(1) - x(2) \\
a_3 &= x(0) + a_1
\end{aligned}
\tag{42}
$$

$$
\begin{aligned}
m_1 &= -\frac{1}{2}\, a_1 \\
m_2 &= -j\frac{\sqrt{3}}{2}\, a_2
\end{aligned}
\tag{43}
$$

$$
\begin{aligned}
c_1 &= x(0) + m_1 \\
X(0) &= a_3 \\
X(1) &= c_1 + m_2 \\
X(2) &= c_1 - m_2
\end{aligned}
\tag{44}
$$

Now, for the algorithm to be used in the nested DFT method, the multiplications by W^0 must be accounted for as explained in Chapter V. Therefore, we wish to minimize the multiplications by W^0 as well. This may be done by modifying the above length 3 DFT as shown below:

$$\begin{aligned} a_1 &= x(1) + x(2) \\ a_2 &= x(1) - x(2) \\ a_3 &= x(0) + a_1 \end{aligned} \tag{45}$$

$$\begin{aligned} m_1 &= (-\tfrac{1}{2} - 1)a_1 = -\tfrac{3}{2}\, a_1 \\ m_2 &= -j\tfrac{\sqrt{3}}{2}\, a_2 \\ m_3 &= W^0\, a_3 = 1\cdot a_3 \end{aligned} \tag{46}$$

$$\begin{aligned} c_1 &= m_3 + m_1 \\ X(0) &= m_3 \\ X(1) &= c_1 + m_2 \\ X(2) &= c_1 - m_2 \end{aligned} \tag{47}$$

The algorithm used for the prime factor FFT has one multiplication , one shift (multiplication by $\frac{1}{2}$), and six additions , as shown in Table 1. The algorithm used for the nested transform has two multiplications, one multiplication by W^0, and six additions. For complex data, the values in Table 1 are only doubled because the coefficients formed from the B$\underline{W}$ portion of the convolution are pure real or pure imaginary numbers. This occurs since the B matrix is such that the W's occur as sums or differences of conjugate pairs.

These short length DFT algorithms may be written in a matrix form as

$$\underline{X} = O\ D\ I\ \underline{x} \tag{48}$$

where I is the μxN matrix representation of the input adds in (42) or (45), D is a μxμ diagonal matrix of the multiplication coefficients in (43) or (46), and O is the Nxμ matrix representation of the output adds in (44) or (47). For our length 3 example in (45)-(47)

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\frac{3}{2} & 0 \\ 0 & 0 & -j\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \end{bmatrix}$$

The summation form of these matrices gives another way to represent these algorithms.

$$X(k) = \sum_{l=0}^{\mu-1} o_{kl}\ d_l \sum_{n=0}^{N-1} i_{ln}\ x(n) \tag{49}$$

In general, $\mu > N$ as shown by the expansion of the block labeled "X" in Figure 1. The input and output additions are indicated by blocks labeled with a "+" .

V. LONG TRANSFORMS FROM SHORT

A. Change to Multidimensions

The short transforms described above can be combined in several ways to provide a long transform of length N. The idea is to convert a one dimensional length $N = M_1 M_2 \cdots M_l$ transform into an l-dimensional transform requiring computation of l shorter, length M_k transforms for $k = 1,2,\cdots,l$. In this paper we use a mapping from one to l dimensions which requires that the M_k factors be relatively prime [11]. Conventional FFT algorithms map one dimension to many dimensions. The Cooley-Tukey algorithm [1] allows common factors in N, while algorithms proposed by I. J. Good [6] and Thomas [7] use a mapping based on the Chinese Remainder Theorem [12] which requires relatively prime factors. We will use the Chinese Remainder mapping which will be described for two factors M_1 and M_2 which are relatively prime.

In the DFT

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W^{nk}$$

the index n of the input sequence x(n) will be called the "input index", and the index k of the output sequence X(k) will be called the "output index". The mapping from one to two dimensions maps the input index n into a pair of indices (n_1, n_2).

$$n_1 = (r_1 n) \bmod M_1 \qquad n_1 = 0, 1, \cdots, M_1-1$$
$$n_2 = (r_2 n) \bmod M_2 \qquad n_2 = 0, 1, \cdots, M_2-1$$

where $r_1 = M_2 \bmod M_1$ and $r_2 = M_1 \bmod M_2$

The output index is similarly mapped into a pair (k_1, k_2).

$$k_1 = k \bmod M_1 \qquad k_1 = 0,1,\cdots,M_1-1$$
$$k_2 = k \bmod M_2 \qquad k_2 = 0,1,\cdots,M_2-1$$

The inverse mapping from two dimensions to one dimension for the output index is

$$k = (s_1 k_1 + s_2 k_2) \bmod N \tag{50}$$

where

$$\begin{aligned} s_1 &= 1 \quad \bmod M_1 \\ s_1 &= 0 \quad \bmod M_2 \end{aligned} \qquad \text{and} \qquad \begin{aligned} s_2 &= 1 \quad \bmod M_2 \\ s_2 &= 0 \quad \bmod M_1 \end{aligned}$$

The inverse mapping for the input index is

$$n = (M_2 n_1 + M_1 n_2) \bmod N \tag{51}$$

When these mappings are used, the DFT becomes

$$X(k_1,k_2) = \sum_{n_1=0}^{M_1-1} \sum_{n_2=0}^{M_2-1} x(n_1,n_2)\, W_{M_2}^{n_2k_2}\, W_{M_1}^{n_1k_1} \tag{52}$$

where $W_{M_1} = e^{-j\frac{2\pi}{M_1}}$ and $W_{M_2} = e^{-j\frac{2\pi}{M_2}}$.

The two dimensional transform in (52) may be implemented by first calculating M_1 length M_2 DFT's,

$$y(n_1,k_2) = \sum_{n_2=0}^{M_2-1} x(n_1,n_2)\, W^{n_2k_2} \tag{53}$$

then calculating M_2 length M_1 DFT's

$$X(k_1,k_2) = \sum_{n_1=0}^{M_1-1} y(n_1,k_2)\, W^{n_1k_1} \tag{54}$$

The short transforms in (53) and (54) can be implemented using convolution methods as in (49). This procedure will be called the prime factor FFT algorithm and is illustrated in Figure 2.

Figure 2 shows a length 15 transform implemented by first calculating five length 3 transforms as in (49), then calculating three length 5 transforms as in (49). Like Figure 1, blocks labeled "+" indicate addition and blocks labeled "X" indicate multiplication in the convolution DFT of (48). Figure 2 is based on similar diagrams in Gold and Rader [13].

Winograd [5] has proposed another implementation of (52) which uses the special structure (49) of the short transforms to nest all multiplications inside of input and output additions. When the length M_1 short transform is written in terms of input additions $i^{(1)}$, output additions $o^{(1)}$, and multiplications $d^{(1)}$ and the length M_2 transform is written in terms of $i^{(2)}$, $o^{(2)}$, and $d^{(2)}$ as in (49), (54) becomes

$$y(n_1,k_2) = \sum_{l=0}^{\mu_2-1} o_{k_2 l}^{(2)}\, d_l^{(2)} \sum_{n_2=0}^{M_2-1} i_{l n_2}^{(2)}\, x(n_1,n_2) \qquad (55)$$

Since $X(k_1,k_2)$ in (52) is a length M_1 transform of $y(n_1,k_2)$ which can also be implemented as in (49), (54) becomes

$$X(k_1,k_2) = \sum_{m=0}^{\mu_1-1} o_{k_1 m}^{(1)}\, d_m^{(1)} \sum_{n_1=0}^{M_1-1} i_{m n_1}^{(1)}\, y(n_1,k_2) \qquad (56)$$

Substituting (55) into (56) we get

$$X(k_1,k_2) = \sum_{m=0}^{\mu_1-1} o_{k_1 m}^{(1)}\, d_m^{(1)} \sum_{n_1=0}^{M_1-1} i_{m n_1}^{(1)} \cdot$$

$$\cdot \sum_{l=0}^{\mu_2-1} o_{k_2 l}^{(2)}\, d_l^{(2)} \sum_{n_2=0}^{M_2-1} i_{l n_2}^{(2)}\, x(n_1,n_2) \qquad (57)$$

The summations in (57) are an explicit representation of the operations indicated in Figure 2. The order of the summation may be changed as shown by Rader and McClellan [9] to nest all multiplications in the center giving

$$X(k_1,k_2) = \sum_{l=0}^{\mu_2-1} o_{k_2 l}^{(2)} \sum_{m=0}^{\mu_1-1} o_{k_1 m}^{(1)}\, d_m^{(1)}\, d_l^{(2)} \cdot$$

$$\cdot \sum_{n_1=0}^{M_1-1} i_{m n_1}^{(1)} \sum_{n_2=0}^{M_2-1} i_{l n_2}^{(2)}\, x(n_1,n_2) \qquad (58)$$

As shown in Figure 3, (58) corresponds to first doing input adds on the rows of $x(n_1,n_2)$, then input adds on the columns. Rows and columns are then multiplied as indicated by $d_m^{(1)}$ and $d_l^{(2)}$. Finally, output additions are performed on columns and rows. This algorithm, proposed by Winograd [5], will be called the nested algorithm in this paper because all multiplications are nested inside of additions as shown in Figure 4.

B. Operation Counts

The number of multiplications required by the nested algorithm is essentially the product of the total number of multiplications for each factor in the DFT, when implemented as in (58). However, some savings may be made for the case where the DFT being computed is at a point by point multiplication involving a product of W^0 coefficients. This corresponds to both d's in (58) being unity. The number of multiplications saved is the product of the number of W^0 multiplications in each short transform. Thus, the equation for the number of multiplications for two factors is

$$\#\text{multiplications} = \mu_1 \mu_2 - \eta_1 \eta_2$$

where η_i = # of multiplications by W^0 for length M_i DFT

μ_i = total # of multiplications for length M_i DFT

α_i = # of additions for length M_i DFT

For three factors we have

$$\#\text{multiplications} = \mu_1 \mu_2 \mu_3 - \eta_1 \eta_2 \eta_3$$

and the pattern continues for more factors.

An equation for the number of additions is based on structures of the nested algorithms in (58) and on Figure 3. The horizontal addition planes in Figure 3 correspond to length M_2 transforms and there are M_1 of them (the indices n_1 and k_1 take on M_1 values). This contributes $M_1 a_2$ additions. The vertical addition planes in Figure 3 correspond to length M_1 transforms and there are μ_2 of them (the index 1 takes on μ_2 values). This contributes $\mu_2 a_1$ additions. Thus for two factors, we have

$$\#\text{additions} = M_1 a_2 + \mu_2 a_1$$

For three factors we have

$$\#\text{additions} = M_1 M_2 a_3 + \mu_3 (M_1 a_2 + \mu_2 a_1)$$

In the same manner, we have with four factors

$$\#\text{additions} = M_1 M_2 M_3 a_4 + \mu_4 \, M_1 M_2 a_3 + \mu_3 (M_1 a_2 + \mu_2 a_1) \tag{59}$$

For the number of multiplications, the ordering of the factors is unimportant. However, the number of additions required depends on the ordering of the factors. For the nested transform additions given in table 2, the best ordering was used and is indicated by the order of the factors. For complex data, the number of real multiplications and real additions is twice the number given by the above equations (multiplications of the complex data are with pure real or pure imaginary coefficients).

In order to compare the operation counts of the nested algorithm with the operations required by the Cooley-Tukey and the prime factor FFT algorithms, we need to look at these latter two algorithms and determine the number of operations they require. In order to make the comparison more realistic, the radix 2 Cooley-Tukey algorithm will perform complex multiplications in three real multiplications and will not count multiplications by W^0 or $\pm j$. For complex data with $N=2^M$ ($\log N = M$), we have [14]:

$$\#\text{multiplications} = 3\left(\frac{N}{2}\log N - \frac{3N}{2} + 2\right)$$

$$\#\text{additions} = 2N\log N + 5(\#\text{multiplications})$$

For the prime factor FFT algorithm we use the special short length transforms intended for this algorithm. (See the first section of Table 1.) With $N = M_1M_2M_3$, M_2M_3 length M_1 transforms are computed, M_1M_3 length M_2 DFT's, and M_1M_2 length M_3 DFT's. We have

$$\#\text{multiplications} = 2(M_2M_3\mu_1 + M_1M_3\mu_2 + M_1M_2\mu_3) \qquad (60)$$

$$\#\text{additions} = 2(M_2M_3\alpha_1 + M_1M_3\alpha_2 + M_1M_2\alpha_3) \qquad (61)$$

for complex data. Using these equations, the three algorithms are compared in Table 2 for several values of N.

Table 2

COMPARISON OF DFT ALGORITHMS

		Nested Algorithm		Prime Factor FFT		Radix 2 FFT	
N	Factors	Multiplies	Adds	Multiplies	Adds	Multiplies	Adds
30	5·3·2	68	384	68	384		
32	2^5					102	830
126	9·7·2	424	3,312	512	2,920		
128	2^7					774	5,662
252	9·7·4	848	7,128	1,024	6,344		
315	9·5·7	1,292	11,286	1,784	8,812		
360	9·5·8	1,152	9,492	1,396	8,708		
504	9·7·8	1,704	15,516	2,300	13,948		
512	2^9					4,614	32,286
840	5·7·8·3	2,592	24,804	4,244	23,172		
1024	2^{10}					10,758	74,270
1260	9·5·7·4	5,168	50,184	7,136	40,288		
2048	2^{11}					24,582	167,966
2520	9·5·7·8	10,344	106,667	15,532	86,876		

VI. COMPARISON OF DFT ALGORITHMS - DISCUSSION OF TABLE 2

Comparisons in Table 2 show that the prime factor FFT algorithm requires from 0% to 64% more multiplications than the nested transform. However, the nested transform requires from 0% to 28% more additions than the prime factor FFT algorithm. In fact, if additions "cost" at least one half as much as multiplications, then the multiply-add cost for the prime factor FFT algorithm is smaller for all lengths shown in Table 2 except for lengths 30 and 840.

To develop an understanding for how these two transforms are related for various choices of factors, we will derive expressions for the number of operations required per output point. From (60), the number of multiplications per output point for the prime factor FFT algorithm is simply the sum of the number of multiplications per point for each factor.

$$\#\text{multiplications/point} = \sum \left(\frac{\mu_i}{M_i} \right) \tag{62}$$

Similarily for the prime factor FFT algorithm,

$$\#\text{additions/point} = \sum \left(\frac{a_i}{M_i} \right) \tag{63}$$

For the nested algorithm, the number of multiplications per point is approximately the product of the number of multiplications per point for each factor.

$$\#\text{multiplications/point} = \prod \left(\frac{\mu_i}{M_i} \right) \tag{64}$$

From (59), the number of additions per point for the nested transform is

$$\#\text{additions/point} = \frac{\alpha_1}{M_1} + \frac{\mu_1}{M_1}\frac{\alpha_2}{M_2} + \frac{\mu_1}{M_1}\frac{\mu_2}{M_2}\frac{\alpha_3}{M_3} + \cdots \tag{65}$$

For the factors used in this study, the number of operations per point is shown in Table 3.

Table 3

	Prime Factor FFT		Nested Algorithm	
N	μ/N	α/N	μ/N	α/N
2	0	1.0	1.0	1.0
3	0.33	2.0	1.0	2.0
4	0	2.0	1.0	2.0
5	0.8	3.4	1.2	3.4
7	1.14	5.14	1.28	5.14
8	0.25	3.25	1.0	3.25
9	0.89	5.44	1.33	5.44

With k factors and an average number of multiplications per point, μ, the nested algorithm requires μ^k multiplications per point. The prime factor FFT algorithm requires $k\mu$ multiplications per point. When $\mu^k > k\mu$ or when $\mu > \sqrt[k-1]{k}$ the prime factor FFT algorithm requires fewer multiplications than the nested algorithm.

Since $\sqrt[k-1]{k}$ becomes smaller with increasing k (more factors) and μ_i increases for the extra factors, which must be large, the prime factor FFT algorithm will have fewer multiplications per point than the nested algorithm when more factors are used.

With k factors and average number of additions per point, α, the prime factor FFT algorithm requires $k\alpha$ additions per point. The nested algorithm requires $\alpha + \mu\alpha + \cdots + \mu^{k-1}\alpha = \frac{\mu^k - 1}{\mu - 1}\alpha$ additions per point for $\mu > 1$. In the special case $N = 2^r \cdot 3 \cdot p$ where p is a prime other than 2 or 3 and r = 1,2, or 3 , both algorithms require the same number of additions since $\mu = 1$ for the factors 2^r and 3. With other factors the prime factor FFT algorithm will have fewer additions. As shown above, the difference in the number of additions will also increase rapidly when more factors are used. This comparison of additions and multiplications per point is further illustrated in Figure 5.

A good strategy would be to use nesting for a few factors until μ began to grow, then combine, using the prime factor FFT algorithm, with another composite length intermediate transform which was done with nesting.

VII. PROGRAMS FOR COMPARING TRANSFORM METHODS

A. Subroutine GOODFT

The prime factor FFT algorithm was used to program a mixed radix DFT in which the short length DFT's are calculated using the fast convolution method previously described. A flow chart of the subroutine GOODFT is shown in Figure 6 and a program listing of the subroutine is given in Appendix B. The input data to be transformed is stored in two length N vectors, XR for the real part and XI for the imaginary part, where N is the length of the DFT to be calculated. N must be a product of at most four mutually prime factors from among the following possible factors: 2,3,4,5,7,8, and 9. If four factors are not used, the unused factors are set equal to one. For example, with N=M1*M2*M3*M4 = 30, we have M1=5, M2=3, M3=2, and M4=1. These factors of one must be the last of the M's. The number of nonunity factors is NFT, which is the number of dimensions in the transform. The prime factor FFT algorithm is described in equations (50) through (54) for the two factor case. This algorithm may be extended to more factors. For example, when the number of mutually prime factors is four, the length N DFT may be calculated as M2*M3*M4 length M1 DFT's, M1*M3*M4 length M2 DFT's, M1*M2*M4 length M3 DFT's and M1*M2*M3 length M4 DFT's.

The first transforms calculated are the length M1

DFT's. For each of the possible combinations of N2, N3, and N4 a length M1 index vector I is calculated using an input mapping

$$n=(M_2M_3M_4n_1+M_1M_3M_4n_2+M_1M_2M_4n_3+M_1M_2M_3n_4)\bmod N \quad (66)$$

The calculation of this index vector and the testing of the values of N2, N3, and N4 are done in the input indexing segments of the subroutine.

The index vectors are used to select the proper data points to be transformed for each of the length M1 DFT's. Thus, when an index vector has been calculated, the proper M1 data points are selected from the length N data vectors XR and XI and stored in temporary vectors UR and UI. A length M1 DFT is then calculated for UR and UI using the fast convolution technique. The results of this transform are stored in UR and UI. Then, the index vector is used once again to transfer the transform results from UR and UI to their correct locations in XR and XI. This selection of M1 data points from the N input data points, the calculation of the M1 point DFT, and the placement of this result into the length N data vector is done in the short transform section of the subroutine.

When all the possible combinations of N2, N3, and N4 have been used, the length M1 DFT's have all been computed. The input indexing portion of the subroutine then reorders the factors so that M4 is now treated as the first factor and the length M4 DFT's are computed. Then, when these are done, M3 and M2 are successively treated as the first factor and the required length M3 and length M2 DFT's are also calculated.

When all of the short transforms for all of the dimensions have been calculated, the vectors XR and XI contain the result of the length N DFT, but in a scrambled order. The unscrambling of the length N transform result is done in the output indexing portion of the subroutine. For the case of four factors, the output index mapping from one to four dimensions is

$$\begin{aligned} k_1 &= k \bmod M_1 \qquad & k_2 &= k \bmod M_2 \\ k_3 &= k \bmod M_3 \qquad & k_4 &= k \bmod M_4 \end{aligned} \tag{67}$$

For a particular value of k, the values of k_1, k_2, k_3, and k_4 are used in (66) for n_1, n_2, n_3, and n_4 to determine the position in the input array of this desired output point. Now, each successive value of k increments all the values of k_1, k_2, k_3, and k_4 by one, starting from zero. Therefore, from (66) we see that the position of each successive output point is located in the input array in the position given by

$$n= k(M_2M_3M_4+M_1M_3M_4+M_1M_2M_4+M_1M_2M_3) \bmod N \tag{68}$$

From (68), we define an output indexing constant,

$$KOUT = (M_2M_3M_4+ M_1M_3M_4+ M_1M_2M_4+ M_1M_2M_3) \bmod N \tag{69}$$

So, we have

$$n = (k \cdot KOUT) \bmod N \qquad (70)$$

The output indexing portion of the subroutine transfers the scrambled results of the length N DFT from XR and XI into A and B, which contain respectively, the real and imaginary parts of the length N DFT in the correct order. After completing this a return is made to the main program.

B. Timing Results

In order to obtain some timing results, the prime factor FFT algorithm was programmed in Fortran and in 8080 microprocessor assembly language. The Fortran prime factor FFT was compared in speed to a mixed radix FFT program written by Singleton [15], which uses a Cooley-Tukey mapping. The FFT subroutine of Singleton is 50% longer than the prime factor FFT subroutine. However, the prime factor FFT uses storage of two complex vectors of length N, while the Singleton FFT subroutine requires one complex length N vector. The results of the time tests for several transform lengths are given in Table 4. These tests were run on an IBM 370 computer for which the ratio of multiply to add time was 3. The power of two algorithm was taken from Rabiner and Gold [14]. It may be 15% slower than an algorithm which stores all powers of W. The timing for the subroutines was accoplished using an interval timer on the IBM 370. The percent saving in time given is the percent by which the Singleton FFT subroutine is slower than the prime factor FFT subroutine. The timing results of Table 4 are for calculating the frequency response of a length 32 finite impulse response digital bandpass filter and are taken from a single run.

Table 4

Time Test on IBM 370/155

Times in Seconds

N	Prime Factor FFT	Singleton's FFT	Radix 2 FFT	% Time Savings
32			0.013	
60	0.017	0.025		47%
64			0.027	
128			0.059	
210	0.080	0.119		49%
256			0.129	
315	0.111	0.179		61%
504	0.168	0.288		71%
512			0.280	
840	0.344	0.509		48%
1024			0.609	
1260	0.540	0.809		50%
2048			1.323	
2520	1.115	1.714		54%

The times for the prime factor FFT may be calculated from the Table 2 values for the total number of operations by the following formula :

$$\text{Time (in milliseconds)} = N(NFT(.052) + .028) + .0096(\#\text{multiplications}) + .0045(\#\text{additions})$$

The input indexing for the program took 52 microseconds per point for each dimension of the transform (NFT equals the number of factors). The output indexing took 28 microseconds per point. The code generated by the Fortran add and multiply statements took 4.5 and 9.6 microseconds,respectively, to run. In the program, the shifts in the short DFT algorithms for the prime factor FFT were done as multiplications.

Next, the 8080 microprocessor assembly language version of the prime factor FFT subroutine was compared in speed with a radix 2 FFT subroutine. The radix 2 FFT was written in assembly language, used three real multiplications for each complex multiplication, and did not multiply by W^0. In addition, the FFT used precalculated values of W_N^i which were stored in a table. The FFT program was much shorter than the prime factor FFT program. The ratio of multiply to add times on the 8080 was approximately 30. A length 252 prime factor FFT requires 3.20 seconds to run. A length 256 radix 2 FFT requires 5.42 seconds to run. So, the radix 2 FFT subroutine is 70% slower than the prime factor FFT subroutine. The savings occur in both the multiplications and additions. The multiplication savings is 80% and the rest is in additions.

VIII. CONCLUSIONS

The conversion of a DFT into a circular convolution leads to new methods for computation of the DFT. For short transforms, these algorithms require few multiplications and additions as shown in Table 1 and as shown in the explicit formulas given in Appendix A.

Long transforms are built up from these short transforms in several ways, which are compared in Table 2. The prime factor FFT algorithm was chosen as the most attractive approach for several reasons. The prime factor FFT algorithm has about the same combined total of multiply-adds as the nested algorithm. However, it is easier to write a general prime factor FFT program. The prime factor FFT can be calculated using less memory than is required for the nested algorithm. It requires less data storage and probably less program memory. Since the prime factor FFT algorithm is done in small pieces, it might run faster on machines with small high speed memory blocks. Special hardware for parallel computation will probably be simpler for the prime factor FFT algorithm.

A general prime factor FFT program was written for an IBM 370 in Fortran and for an 8080 microprocessor in assembly language. The running time for this new algorithm was compared with a conventional FFT. In the 370 comparison the new algorithm was compared with the mixed radix algorithm of Singleton [15], since the prime factor FFT algorithm is a mixed radix algorithm. A reduction of approximately 50% was observed (see Table 4). Much larger savings may be expected if special hardware is constructed for the short convolution-based algorithms.

Many open questions remain. How should one combine nesting and prime factor FFT techniques to obtain long transforms from short ones? Should the multidimensional expansion always be done at the transform level, or should the convolutions contained within transforms also be implemented in multidimensional expansions? How can one improve on the indexing schemes required for these new transforms? It is likely that continuing development of longer and more efficient convolution algorithms will make implementations of the DFT using convolution even more attractive.

REFERENCES

1. J.W. Cooley and J.W. Tukey, "An Algorithm for the machine calculation of complex Fourier series," Math. of comput., vol. 19, pp 297-301, April 1965.

2. C. Rader, "Discrete Fourier Transforms When the Number of Data Samples is Prime" Proceedings of the IEEE, vol. 56, pp 1107-1108, June 1968.

3. S. Winograd, "Some Bilinear Forms Whose Multiplicative Complexity Depends on the Field of Constants," IBM Research Report, RC 5669, IBM Watson Research Center, P.O. Box 218, Yorktown Hts. New York, 10598, Oct.10,1975

4. R. Agarwal and J.W. Cooley, "New Algorithms for Digital Convolution," presented at Arden House Workshop, Feb. 1976 and to appear as IBM Report.

5. S. Winograd, "On Computing the Discrete Fourier Transform," Proc. Nat. Acad. Sci. USA, Vol. 73, No. 4, pp 1005-1006, April 1976.

6. I.J. Good, "The Interaction Algorithm and Practical Fourier Analysis," J. Royal Statist. Soc., Ser. B. Vol. 20, pp 361-372, 1958, Addendum, Vol. 22, 1960 pp 372-375, (MR 21 1674; MR 23 A4231).

7. L.H. Thomas, "Using a Computer to Solve Problems in Physics," Applications of Digital Computers, Ginn and Co., Boston, Mass., 1963.

8. Mostow, C.D., Sampson, J.H., and Meyer, J., Fundamental Structures of Algebra, McGraw Hill Book Co., N.Y. 1963.

9. J. McClellan and C.M. Rader, "There is Something Much Faster than the Fast Fourier Transform," Notes from Seminar, Oct. 21, 1976.

10. H.F. Silverman, "An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA)", to appear in IEEE Transactions on Acoustics, Speech, and Signal Processing.

11. C.S. Burrus, "Index Mappings for Multidimensional Formulation of the DFT and Convolution," to appear in IEEE Transactions on Acoustics, Speech, and Signal Processing.

12. D.E. Knuth, "Seminumerical Algorithms," Vol. 2 of <u>The Art of Computer Programming</u>, Addison-Wesley, Reading, Mass., 1971.

13. B. Gold and C.M. Rader, <u>Digital Processing of Signals</u>, McGraw Hill Book Company, New York, 1969, pp 194.

14. Rabiner, L.R., and Gold, B., <u>Theory and Application of Digital Signal Processing</u>, Prentice Hall Inc., Englewood Cliffs, N.J. 1975.

15. R.C. Singleton, "An Algorithm for Computing the Mixed Radix Fast Fourier Transform," IEEE Trans. Audio Electroacoust., Vol. AU-17, pp93-103, June 1969.

APPENDIX A.

3 POINT DFT ALGORITHM

$a_1 = x(1) + x(2)$

$a_2 = x(1) - x(2)$

$a_3 = x(0) + a_1$

$m_1 = \frac{1}{2} a_1$

$m_2 = 0.86603\ a_2$

$c_1 = x(0) - m_1$

$X(0) = a_3$

$X(1) = c_1 - jm_2$

$X(2) = c_1 + jm_2$

1 multiplication

1 shift

6 additions

5 POINT DFT ALGORITHM

$a_1 = x(1) + x(4)$

$a_2 = x(1) - x(4)$

$a_3 = x(2) + x(3)$

$a_4 = x(2) - x(3)$

$a_5 = a_2 + a_4$

$a_6 = a_1 - a_3$

$a_7 = a_1 + a_3$

$a_8 = x(0) + a_7$

$m_1 = 0.95106\ a_5$

$m_2 = 1.53884\ a_2$

$m_3 = 0.36327\ a_4$

$m_4 = 0.55902\ a_6$

$m_5 = \frac{1}{4} a_7$

$c_1 = x(0) - m_5$

$c_2 = c_1 + m_4$

$c_3 = c_1 - m_4$

$c_4 = m_1 - m_3$

$c_5 = m_2 - m_1$

$X(0) = a_8$

$X(1) = c_2 - jc_4$

$X(2) = c_3 - jc_5$

$X(3) = c_3 + jc_5$

$X(4) = c_2 + jc_4$

4 multiplications

2 shifts

17 additions

7 POINT DFT ALGORITHM

$a_1 = x(1) + x(6)$

$a_2 = x(1) - x(6)$

$a_3 = x(2) + x(5)$

$a_4 = x(2) - x(5)$

$a_5 = x(3) + x(4)$

$a_6 = x(3) - x(4)$

$a_7 = a_1 + a_3 + a_5$

$a_8 = a_1 - a_5$

$a_9 = -a_3 + a_5$

$a_{10} = -a_1 + a_3$

$a_{11} = a_2 + a_4 - a_6$

$a_{12} = a_2 + a_6$

$a_{13} = -a_4 - a_6$

$a_{14} = -a_2 + a_4$

$a_{15} = x(0) + a_7$

$m_1 = 0.16667\ a_7$

$m_2 = 0.79016\ a_8$

$m_3 = 0.05585\ a_9$

$m_4 = 0.73430\ a_{10}$

$m_5 = 0.44096\ a_{11}$

$m_6 = 0.34087\ a_{12}$

$m_7 = 0.53397\ a_{13}$

$m_8 = 0.87484\ a_{14}$

$c_1 = x(0) - m_1$

$c_2 = c_1 + m_2 + m_3$

$c_3 = c_1 - m_2 - m_4$

$c_4 = c_1 - m_3 + m_4$

$c_5 = m_5 + m_6 - m_7$

$c_6 = m_5 - m_6 - m_8$

$c_7 = -m_5 - m_7 - m_8$

$X(0) = a_{15}$

$X(1) = c_2 - jc_5$

$X(2) = c_3 - jc_6$

$X(3) = c_4 - jc_7$

$X(4) = c_4 + jc_7$

$X(5) = c_3 + jc_6$

$X(6) = c_2 + jc_5$

8 multiplications

36 additions

9 POINT DFT ALGORITHM

$a_1 = x(1) + x(8)$

$a_2 = x(1) - x(8)$

$a_3 = x(2) + x(7)$

$a_4 = x(2) - x(7)$

$a_5 = x(4) + x(5)$

$a_6 = x(4) - x(5)$

$a_7 = x(3) + x(6)$

$a_8 = x(3) - x(6)$

$a_9 = -a_1 + a_5$

$a_{10} = a_1 - a_3$

$a_{11} = -a_3 + a_5$

$a_{12} = a_2 - a_6$

$a_{13} = a_2 + a_4$

$a_{14} = -a_4 - a_6$

$a_{15} = a_1 + a_3 + a_5$

$a_{16} = a_2 - a_4 + a_6$

$a_{17} = x(0) + a_{15} + a_7$

$m_1 = 0.19740\ a_9$

$m_2 = 0.56858\ a_{10}$

$m_3 = 0.37111\ a_{11}$

$m_4 = 0.54253\ a_{12}$

$m_5 = 0.10026\ a_{13}$

$m_6 = 0.44228\ a_{14}$

$m_7 = \frac{1}{2}\ a_7$

$m_8 = 0.86603\ a_8$

$m_9 = \frac{1}{2}\ a_{15}$

$m_{10} = 0.86603\ a_{16}$

$c_1 = x(0) - m_7$

$c_2 = m_2 - m_3$

$c_3 = m_1 + m_3$

$c_4 = m_1 + m_2$

$c_5 = c_1 + c_2 - c_3$

$c_6 = c_1 + c_3 + c_4$

$c_7 = c_1 - c_2 - c_4$

$c_8 = m_4 - m_6$

$c_9 = m_5 - m_6$

$c_{10} = m_4 - m_5$

$c_{11} = c_8 + c_9 + m_8$

$c_{12} = c_8 + c_{10} - m_8$

$c_{13} = -c_9 + c_{10} + m_8$

$c_{14} = x(0) + a_7 - m_9$

$X(0) = a_{17}$

$X(1) = c_5 - jc_{11}$

$X(2) = c_6 - jc_{12}$

$X(3) = c_{14} - jm_{10}$

$X(4) = c_7 - jc_{13}$

$X(5) = c_7 + jc_{13}$

$X(6) = c_{14} + jm_{10}$

$X(7) = c_6 + jc_{12}$

$X(8) = c_5 + jc_{11}$

8 multiplications

2 shifts

49 additions

APPENDIX B.

Prime Factor FFT Program Listing

```
      SUBROUTINE GOODFT(XR,XI,N,M1,M2,M3,M4,NFT,KOUT,A,B)
C  THE SUBROUTINE GOODFT COMPUTES A LENGTH N DFT OF THE
C  INPUT DATA WHICH IS IN TWO VECTORS, XR THE REAL PART AND
C  XI THE IMAGINARY PART. BOTH XR AND XI ARE LENGTH N VEC-
C  TORS. THE LENGTH OF THE DFT, N, MUST BE A PRODUCT OF AT
C  MOST FOUR MUTUALLY PRIME FACTORS. THE POSSIBLE FACTORS
C  ARE 2,3,4,5,7,8, AND 9. THESE FACTORS ARE M1, M2, M3, AND
C  M4. IF THE FOUR FACTORS ARE NOT ALL USED, THE UNUSED
C  FACTORS ARE SET EQUAL TO 1. FOR EXAMPLE WITH N=30, WE
C  HAVE M1=5, M2=3, M3=2, AND M4=1. THE FACTORS OF ONE MUST
C  BE THE LAST OF THE M'S. THE NUMBER OF NONUNITY FACTORS IS
C  NFT. KOUT IS AN OUTPUT INDEXING CONSTANT WHICH IS PRE-
C  COMPUTED. KOUT = (K1+K2+K3+K4)MOD N WHERE K1=M2*M3*M4,
C  K2=M1*M3*M4, K3=M1*M2*M4, K4=M1*M2*M3, AND K2=0 IF M2=1,
C  K3=0 IF M3=1, AND K4=0 IF M4=1. FOR EXAMPLE, N=30, K1=6,
C  K2=10, K3=15, K4=0 AND KOUT=31 MOD 30 = 1. THE TRANSFORMED
C  RESULT IS STORED IN TWO LENGTH N VECTORS, A AND B. A CON-
C  TAINS THE REAL PART AND B CONTAINS THE IMAGINARY PART OF
C  THE RESULT. THE DFT COMPUTED BY THIS SUBROUTINE USES A
C  POSITIVE EXPONENT FOR W. IE W=EXP(J*2*PI/N).
C  DEAN KOLBA, JULY 1976.
      DIMENSION XR(2520),XI(2520),A(2520),B(2520)
      DIMENSION UR(9),UI(9),I(9)
      REAL MR1,MR2,MR3,MR4,MR5,MR6,MR7,MR8,MR9,MR10
      REAL MI1,MI2,MI3,MI4,MI5,MI6,MI7,MI8,MI9,MI10
      NF=NFT
C  ORDER FACTORS FOR TRANSFORMS OF LENGTH M1
      MM1=M1
      MM2=M2
      MM3=M3
      MM4=M4
      GOTO 20
   10 GOTO(12,13,14),NF
C  ORDER FACTORS FOR TRANSFORMS OF LENGTH M2
   12 MM1=M2
      MM2=M1
      MM3=M3
      MM4=M4
      GOTO 20
C  ORDER FACTORS FOR TRANSFORMS OF LENGTH M3
   13 MM1=M3
      MM2=M1
      MM3=M2
      MM4=M4
      GOTO 20
```

```
C   ORDER FACTORS FOR TRANSFORMS OF LENGTH M4
    14 MM1=M4
       MM2=M1
       MM3=M2
       MM4=M3
C   INDEXING INITIALIZATION FOR THE TRANSFORMS
    20 N2=0
       N3=0
       N4=0
       K1=MM2*MM3*MM4
       K2=MM1*MM3*MM4
       K3=MM1*MM2*MM4
       K4=MM1*MM2*MM3
       I(1)=0
C   INPUT INDEXING ALONG ONE DIMENSION
    21 DO 22 J=2,MM1
       I(J)=I(J-1)+K1
       IF(I(J) .LT. N) GOTO 22
       I(J)=I(J)-N
    22 CONTINUE
C   TRANSFERRING DATA TO TEMPORARY VECTORS UR AND UI
    30 DO 31 J=1,MM1
       IJ=I(J)+1
       UR(J)=XR(IJ)
    31 UI(J)=XI(IJ)
C   TRANSFORM UR, UI
       GOTO(50,200,300,400,500,50,700,800,900),MM1
C   PLACE RESULT OF TRANSFORM BACK IN XR AND XI
    40 DO 41 J=1,MM1
       IJ=I(J)+1
       XR(IJ)=UR(J)
    41 XI(IJ)=UI(J)
C   TESTING FOR COMPLETION OF THIS FACTOR'S TRANSFORMS
       IF(N2 .NE. MM2-1)GOTO 51
       N2=0
       IF(N3 .NE. MM3-1)GOTO 52
       N3=0
       IF(N4 .NE. MM4-1)GOTO 53
    50 NF=NF-1
       IF(NF.EQ.0)GOTO1000
       GOTO 10
C   INPUT INDEXING ALONG OTHER DIMENSIONS
    51 N2=N2+1
       DO 54 J=1,MM1
       I(J)=I(J)+K2
       IF(I(J) .LT. N)GOTO 54
       I(J)=I(J)-N
    54 CONTINUE
       GOTO 30
    52 N3=N3+1
       I(1)=K3*N3+K4*N4
       IF(I(1).LT.N)GOTO 21
       I(1)=I(1)-N
       GOTO 21
```

```
   53 N4=N4+1
      I(1)=K4*N4
      GOTO 21
C  UNSCRAMBLING TRANSFORM RESULT
 1000 II=1
      J=1
      GOTO 1001
 1002 IF(J .GT. N)GOTO 1003
      II=II+KOUT
 1004 IF(II .LE. N)GOTO 1001
      II=II-N
      GOTO 1004
 1001 A(J)=XR(II)
      B(J)=XI(II)
      J=J+1
      GOTO 1002
C  2 POINT TRANSFORM
  200 URX=UR(1)+UR(2)
      UIX=UI(1)+UI(2)
      UR(2)=UR(1)-UR(2)
      UI(2)=UI(1)-UI(2)
      UR(1)=URX
      UI(1)=UIX
      GOTO 40
C  3 POINT TRANSFORM
  300 AR=UR(2)+UR(3)
      AI=UI(2)+UI(3)
      MR1=-1.5*AR
      MI1=-1.5*AI
      MR2=0.8660254*(UR(2)-UR(3))
      MI2=0.8660254*(UI(2)-UI(3))
      UR(1)=AR+UR(1)
      UI(1)=AI+UI(1)
      MR1=UR(1)+MR1
      MI1=UI(1)+MI1
      UR(2)=MR1-MI2
      UI(2)=MI1+MR2
      UR(3)=MR1+MI2
      UI(3)=MI1-MR2
      GOTO 40
C  4 POINT TRANSFORM
  400 AR1=UR(1)+UR(3)
      AI1=UI(1)+JI(3)
      AR2=UR(1)-UR(3)
      AI2=UI(1)-UI(3)
      AR3=UR(2)+UR(4)
      AI3=UI(2)+UI(4)
      AR4=UR(2)-UR(4)
      AI4=UI(2)-UI(4)
      UR(1)=AR1+AR3
      UI(1)=AI1+AI3
      UR(2)=AR2-AI4
      UI(2)=AI2+AR4
      UR(3)=AR1-AR3
```

```
      UI(3)=AI1-AI3
      UR(4)=AR2+AI4
      UI(4)=AI2-AR4
      GOTO 40
C   5 POINT TRANSFORM
  500 AR1=UR(2)+UR(5)
      AI1=UI(2)+UI(5)
      AR2=UR(2)-UR(5)
      AI2=UI(2)-UI(5)
      AR3=UR(3)+UR(4)
      AI3=UI(3)+UI(4)
      AR4=UR(3)-UR(4)
      AI4=UI(3)-UI(4)
      AR5=AR1+AR3
      AI5=AI1+AI3
      MR1=0.95105652*(AR2+AR4)
      MI1=0.95105652*(AI2+AI4)
      MR2=1.5388418*AR2
      MI2=1.5388418*AI2
      MR3=0.36327126*AR4
      MI3=0.36327126*AI4
      MR4=0.55901699*(AR1-AR3)
      MI4=0.55901699*(AI1-AI3)
      MR5=-1.25*AR5
      MI5=-1.25*AI5
      UR(1)=UR(1)+AR5
      UI(1)=UI(1)+AI5
      MR5=UR(1)+MR5
      MI5=UI(1)+MI5
      AR1=MR5+MR4
      AI1=MI5+MI4
      AR2=MR5-MR4
      AI2=MI5-MI4
      AR3=MR1-MR3
      AI3=MI1-MI3
      AR4=MR1-MR2
      AI4=MI1-MI2
      UR(2)=AR1-AI3
      UI(2)=AI1+AR3
      UR(3)=AR2+AI4
      UI(3)=AI2-AR4
      UR(4)=AR2-AI4
      UI(4)=AI2+AR4
      UR(5)=AR1+AI3
      UI(5)=AI1-AR3
      GOTO 40
C   7 POINT TRANSFORM
  700 AR1=UR(2)+UR(7)
      AI1=UI(2)+UI(7)
      AR2=UR(2)-UR(7)
      AI2=UI(2)-UI(7)
      AR3=UR(3)+UR(6)
      AI3=UI(3)+UI(6)
      AR4=UR(3)-UR(6)
```

```
AI4=UI(3)-UI(6)
AR5=UR(4)+UR(5)
AI5=UI(4)+UI(5)
AR6=UR(4)-UR(5)
AI6=UI(4)-UI(5)
AR7=AR1+AR3+AR5
AI7=AI1+AI3+AI5
MR1=-1.1666667*AR7
MI1=-1.1666667*AI7
MR2=0.79015647*(AR1-AR5)
MI2=0.79015647*(AI1-AI5)
MR3=0.055854267*(AR5-AR3)
MI3=0.055854267*(AI5-AI3)
MR4=0.7343022*(AR3-AR1)
MI4=0.7343022*(AI3-AI1)
MR5=0.44095855*(AR2+AR4-AR6)
MI5=0.44095855*(AI2+AI4-AI6)
MR6=0.34087293*(AR2+AR6)
MI6=0.34087293*(AI2+AI6)
MR7=-0.53396936*(-AR6-AR4)
MI7=-0.53396936*(-AI6-AI4)
MR8=0.87484229*(AR4-AR2)
MI8=0.87484229*(AI4-AI2)
UR(1)=UR(1)+AR7
UI(1)=UI(1)+AI7
AR1=UR(1)+MR1
AI1=UI(1)+MI1
AR2=AR1+MR2+MR3
AI2=AI1+MI2+MI3
AR3=AR1-MR2-MR4
AI3=AI1-MI2-MI4
AR4=AR1-MR3+MR4
AI4=AI1-MI3+MI4
AR5=MR5+MR6+MR7
AI5=MI5+MI6+MI7
AR6=MR5-MR6-MR8
AI6=MI5-MI6-MI8
AR7=MR5-MR7+MR8
AI7=MI5-MI7+MI8
UR(2)=AR2-AI5
UI(2)=AI2+AR5
UR(3)=AR3-AI6
UI(3)=AI3+AR6
UR(4)=AR4+AI7
UI(4)=AI4-AR7
UR(5)=AR4-AI7
UI(5)=AI4+AR7
UR(6)=AR3+AI6
UI(6)=AI3-AR6
UR(7)=AR2+AI5
UI(7)=AI2-AR5
GOTO 40
```

```
C   8 POINT TRANSFORM
  800 AR1=UR(2)-UR(8)
      AI1=UI(2)-UI(8)
      AR2=UR(2)+UR(8)
      AI2=UI(2)+UI(8)
      AR3=UR(4)-UR(6)
      AI3=UI(4)-UI(6)
      AR4=UR(4)+UR(6)
      AI4=UI(4)+UI(6)
      AR5=UR(1)-UR(5)
      AI5=UI(1)-JI(5)
      AR6=UR(1)+UR(5)
      AI6=UI(1)+UI(5)
      AR7=UR(3)-UR(7)
      AI7=UI(3)-UI(7)
      AR8=UR(3)+UR(7)
      AI8=UI(3)+UI(7)
      MR1=0.70710678*(AR1+AR3)
      MI1=0.70710678*(AI1+AI3)
      MR2=0.70710678*(AR2-AR4)
      MI2=0.70710678*(AI2-AI4)
      MR3=AR2+AR4
      MI3=AI2+AI4
      MR4=AR6+AR8
      MI4=AI6+AI8
      MR5=AR6-AR8
      MI5=AI6-AI8
      MR6=AR1-AR3
      MI6=AI1-AI3
      MR7=AR5+MR2
      MI7=AI5+MI2
      MR8=AR5-MR2
      MI8=AI5-MI2
      MR9=AR7+MR1
      MI9=AI7+MI1
      MR10=AR7-MR1
      MI10=AI7-MI1
      UR(1)=MR4+MR3
      UI(1)=MI4+MI3
      UR(2)=MR7-MI9
      UI(2)=MI7+MR9
      UR(3)=MR5-MI6
      UI(3)=MI5+MR6
      UR(4)=MR8+MI10
      UI(4)=MI8-MR10
      UR(5)=MR4-MR3
      UI(5)=MI4-MI3
      UR(6)=MR8-MI10
      UI(6)=MI8+MR10
      UR(7)=MR5+MI6
      UI(7)=MI5-MR6
      UR(8)=MR7+MI9
      UI(8)=MI7-MR9
      GOTO 40
```

```
C   9 POINT TRANSFORM
  900 AR1=UR(2)+UR(9)
      AI1=UI(2)+UI(9)
      AR2=UR(2)-UR(9)
      AI2=UI(2)-UI(9)
      AR3=UR(3)+UR(8)
      AI3=UI(3)+UI(8)
      AR4=UR(3)-UR(8)
      AI4=UI(3)-UI(8)
      AR5=UR(5)+UR(6)
      AI5=UI(5)+UI(6)
      AR6=UR(5)-UR(6)
      AI6=UI(5)-UI(6)
      AR7=UR(4)+UR(7)
      AI7=UI(4)+UI(7)
      AR8=UR(4)-UR(7)
      AI8=UI(4)-UI(7)
      AR=AR1+AR3+AR5
      AI=AI1+AI3+AI5
      MR1=-0.5*AR7
      MI1=-0.5*AI7
      MR2=0.8660254*AR8
      MI2=0.8660254*AI8
      MR3=0.19746542*(-AR1+AR5)
      MI3=0.19746542*(-AI1+AI5)
      MR4=0.56857902*(AR1-AR3)
      MI4=0.56857902*(AI1-AI3)
      MR5=0.3711136*(-AR3+AR5)
      MI5=0.3711136*(-AI3+AI5)
      MR6=0.54253179*(AR2-AR6)
      MI6=0.54253179*(AI2-AI6)
      MR7=0.10025582*(AR2+AR4)
      MI7=0.10025582*(AI2+AI4)
      MR8=0.44227597*(-AR4-AR6)
      MI8=0.44227597*(-AI4-AI6)
      MR9=-1.5*AR
      MI9=-1.5*AI
      MR10=0.8660254*(AR2-AR4+AR6)
      MI10=0.8660254*(AI2-AI4+AI6)
      AR1=UR(1)+MR1
      AI1=UI(1)+MI1
      UR(1)=AR+AR7+UR(1)
      UI(1)=AI+AI7+UI(1)
      AR=UR(1)+MR9
      AI=UI(1)+MI9
      AR2=MR4-MR5
      AI2=MI4-MI5
      AR3=MR3+MR4
      AI3=MI3+MI4
      AR4=MR7-MR8
      AI4=MI7-MI8
      AR5=MR6-MR7
      AI5=MI6-MI7
      AR6=AR2-MR5-MR3+AR1
      AI6=AI2-MI5-MI3+AI1
```

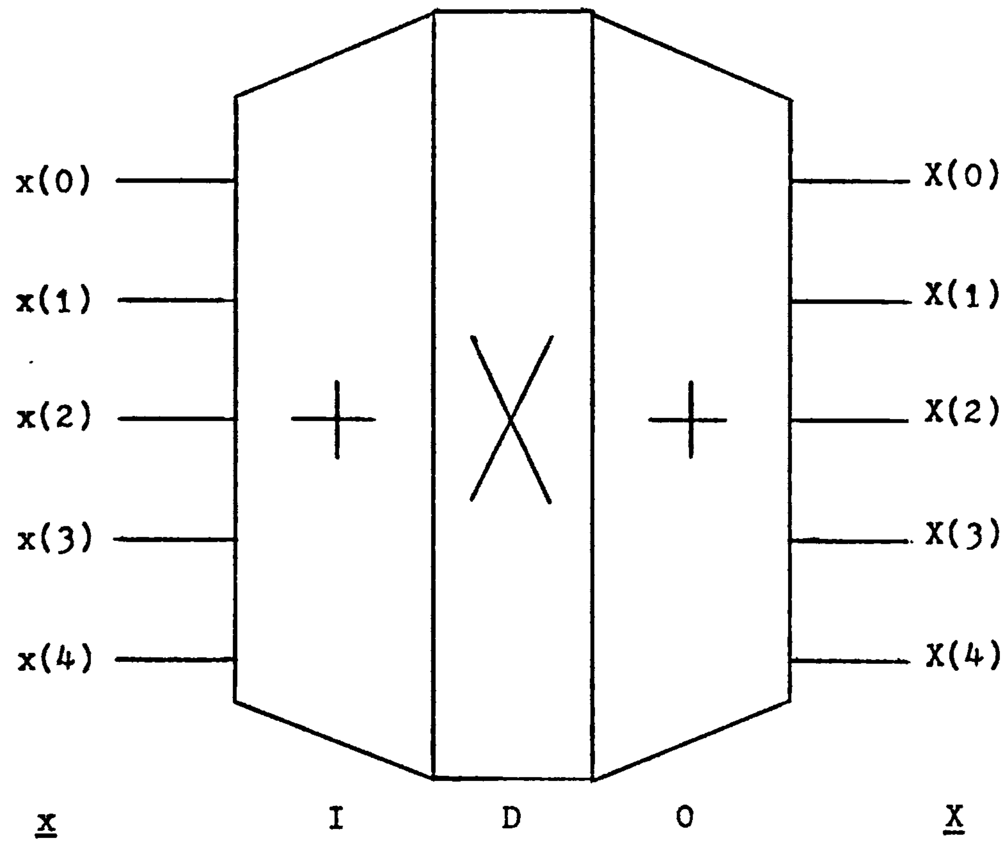


Figure 1. DFT implemented with convolution showing expansion caused by more than 1 multiply per point.

```
      AR7=AR3+MR3+MR5+AR1
      AI7=AI3+MI3+MI5+AI1
      AR8=-AR3-AR2+AR1
      AI8=-AI3-AI2+AI1
      MR1=MR6-MR8
      MI1=MI6-MI8
      MR3=AR4+MR1+MR2
      MI3=AI4+MI1+MI2
      MR4=AR5+MR1-MR2
      MI4=AI5+MI1-MI2
      MR5=AR5-AR4+MR2
      MI5=AI5-AI4+MI2
      UR(2)=AR6-MI3
      UI(2)=AI6+MR3
      UR(3)=AR7-MI4
      UI(3)=AI7+MR4
      UR(4)=AR-MI10
      UI(4)=AI+MR10
      UR(5)=AR8-MI5
      UI(5)=AI8+MR5
      UR(6)=AR8+MI5
      UI(6)=AI8-MR5
      UR(7)=AR+MI10
      UI(7)=AI-MR10
      UR(8)=AR7+MI4
      UI(8)=AI7-MR4
      UR(9)=AR6+MI3
      UI(9)=AI6-MR3
      GOTO 40
 1003 RETURN
      END
```

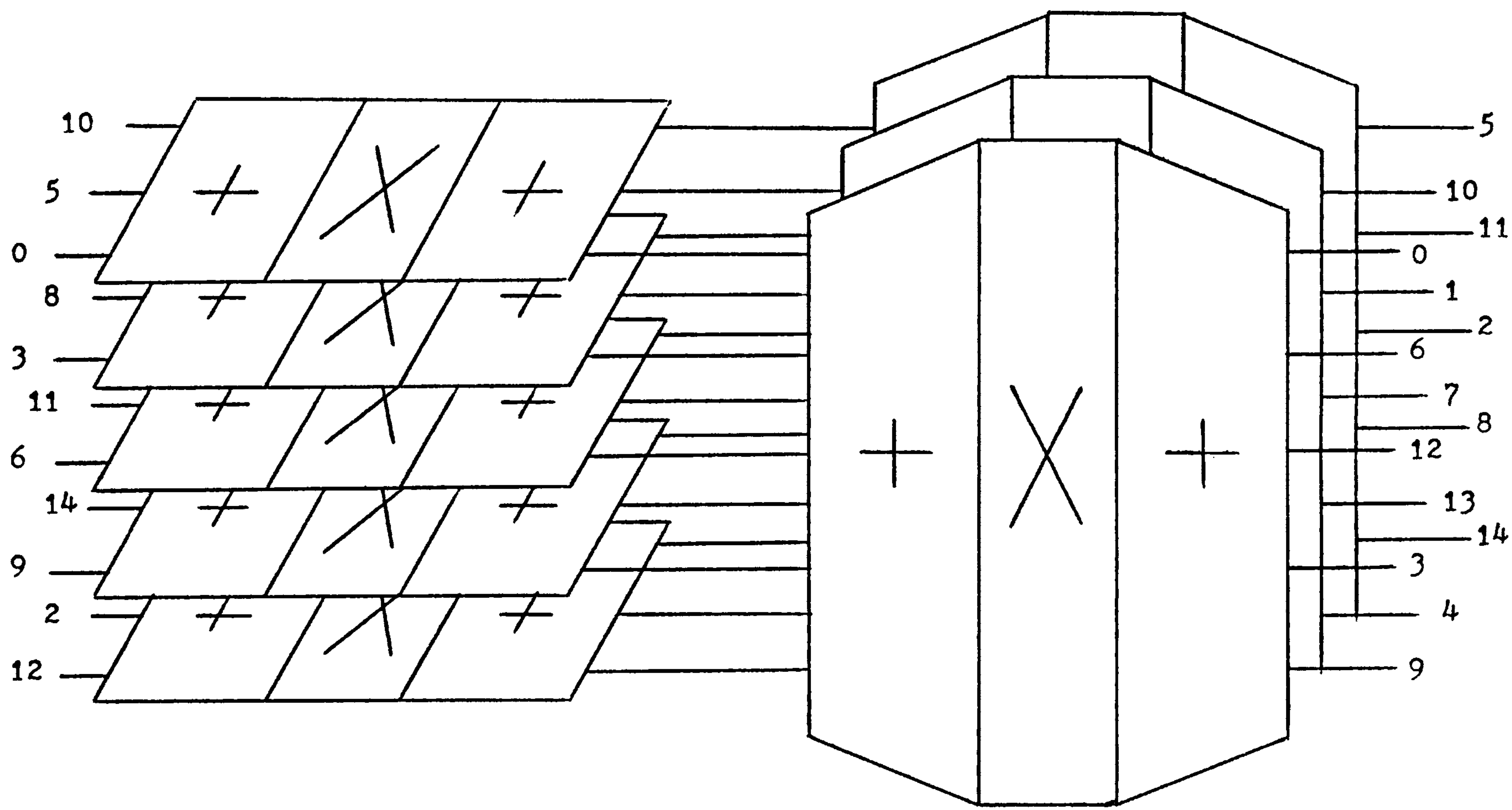


Figure 2. Multidimensional prime factor FFT algorithm for length 15.

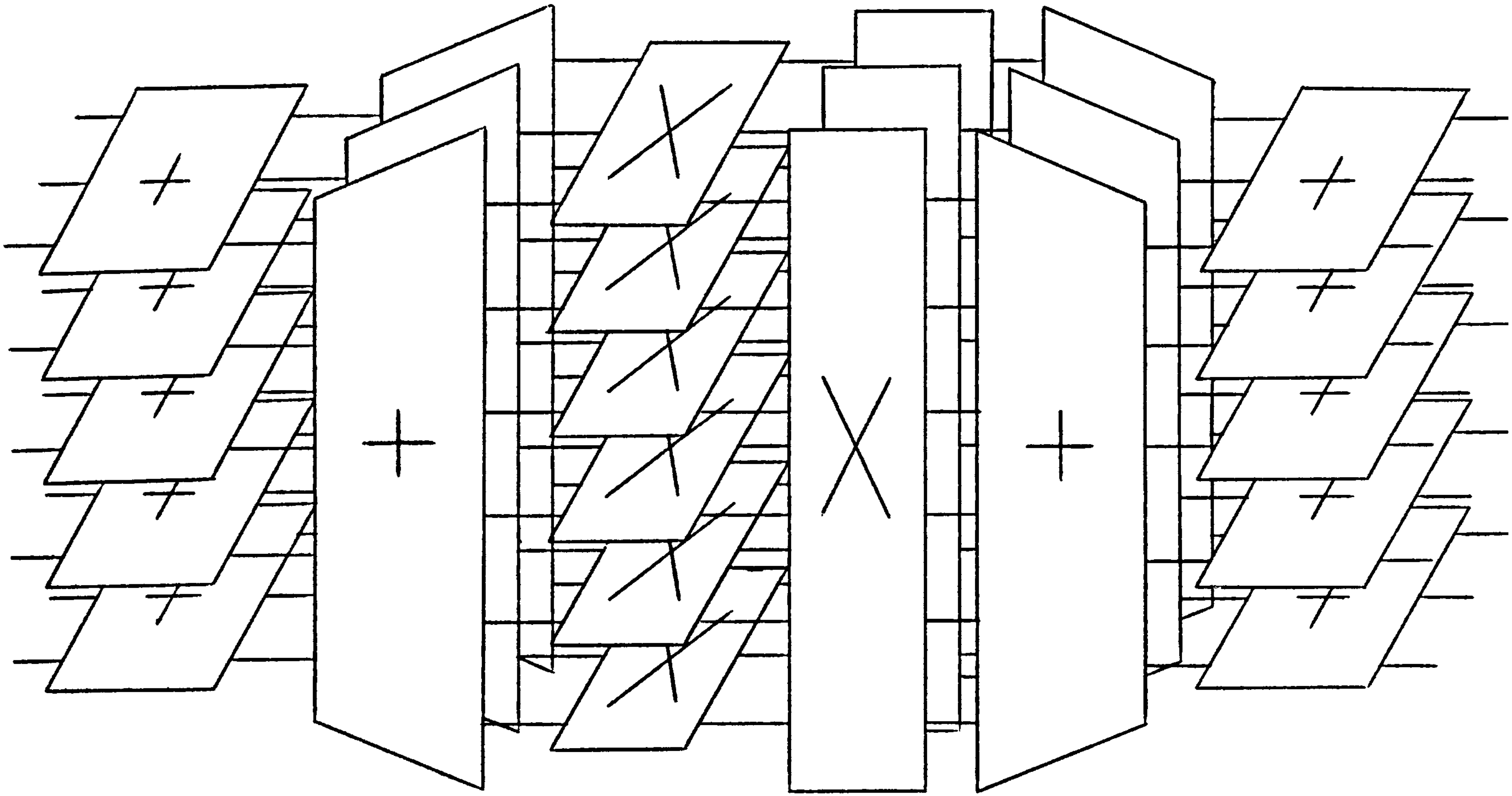


Figure 3. Rearrangement of operations to nest multiplications inside of additions.

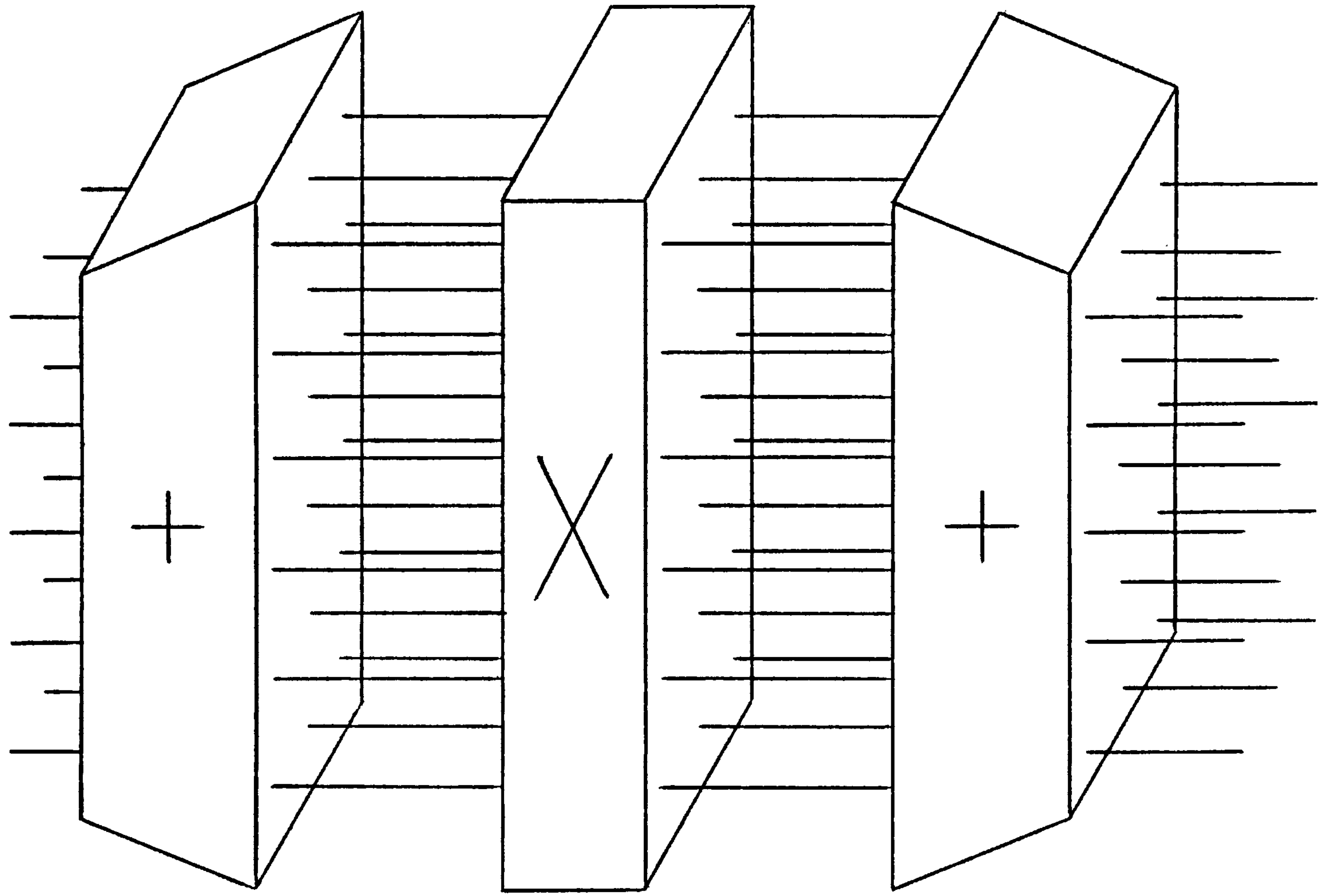


Figure 4. Length 15 DFT implemented by the nested algorithm.

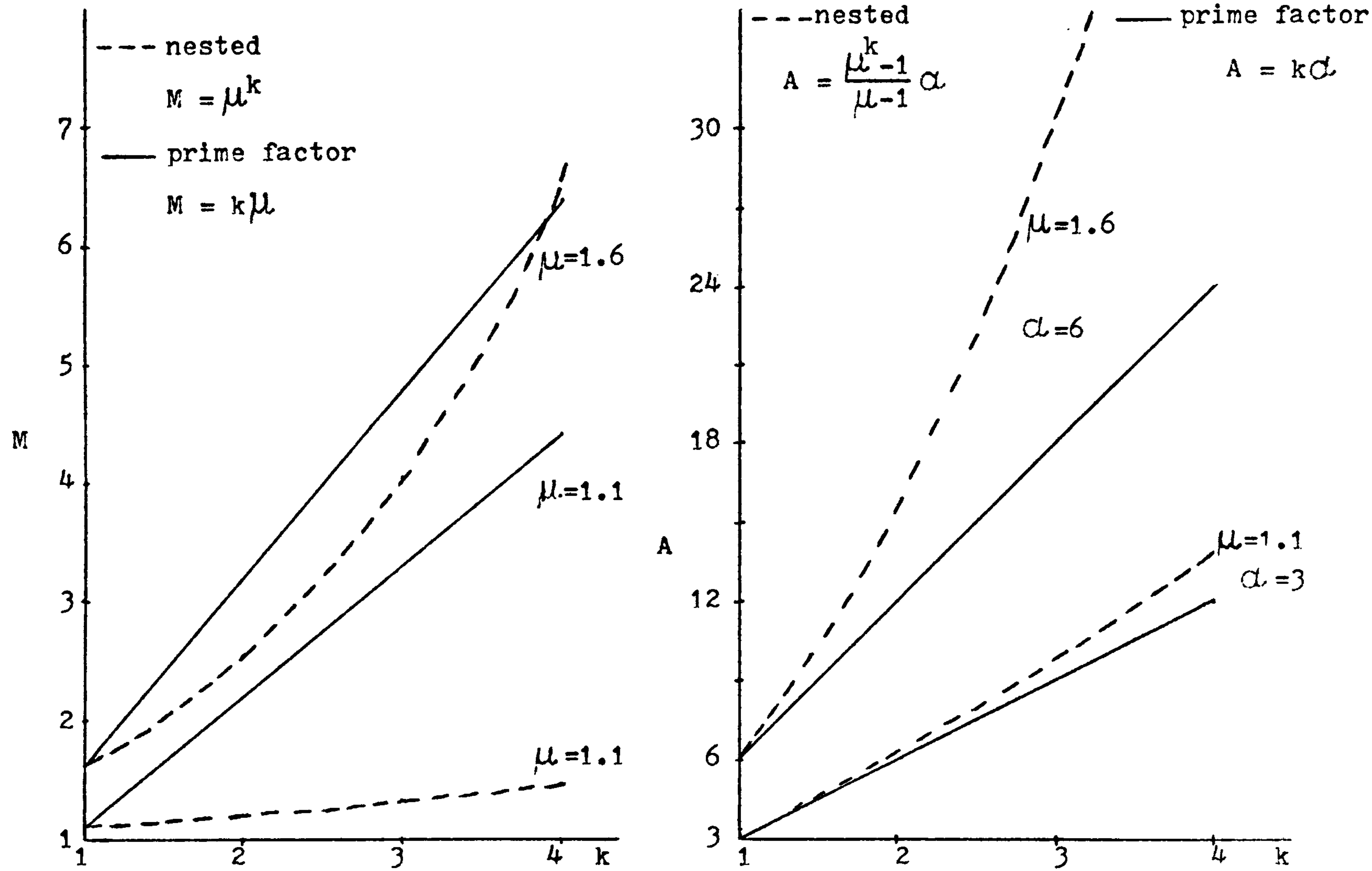
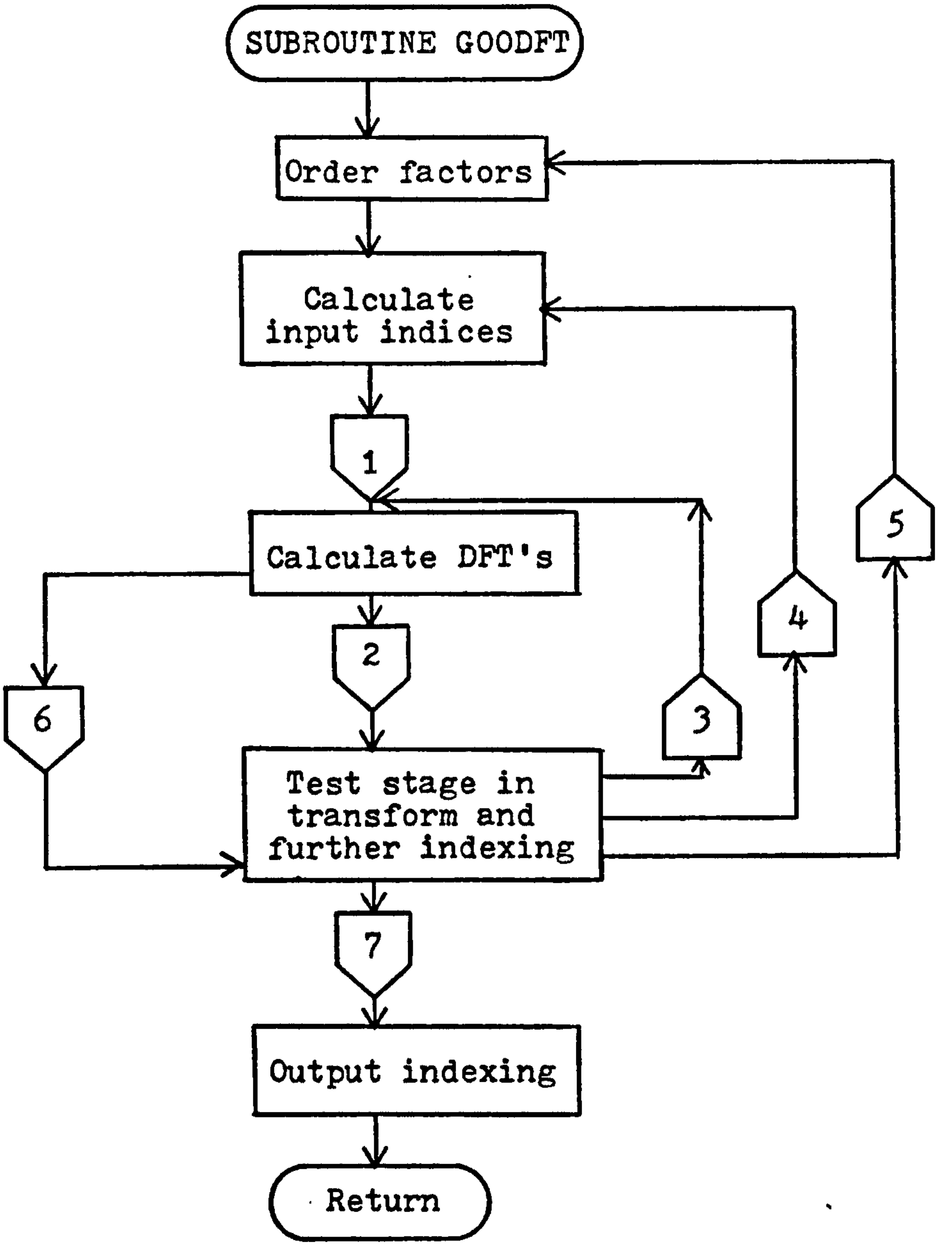


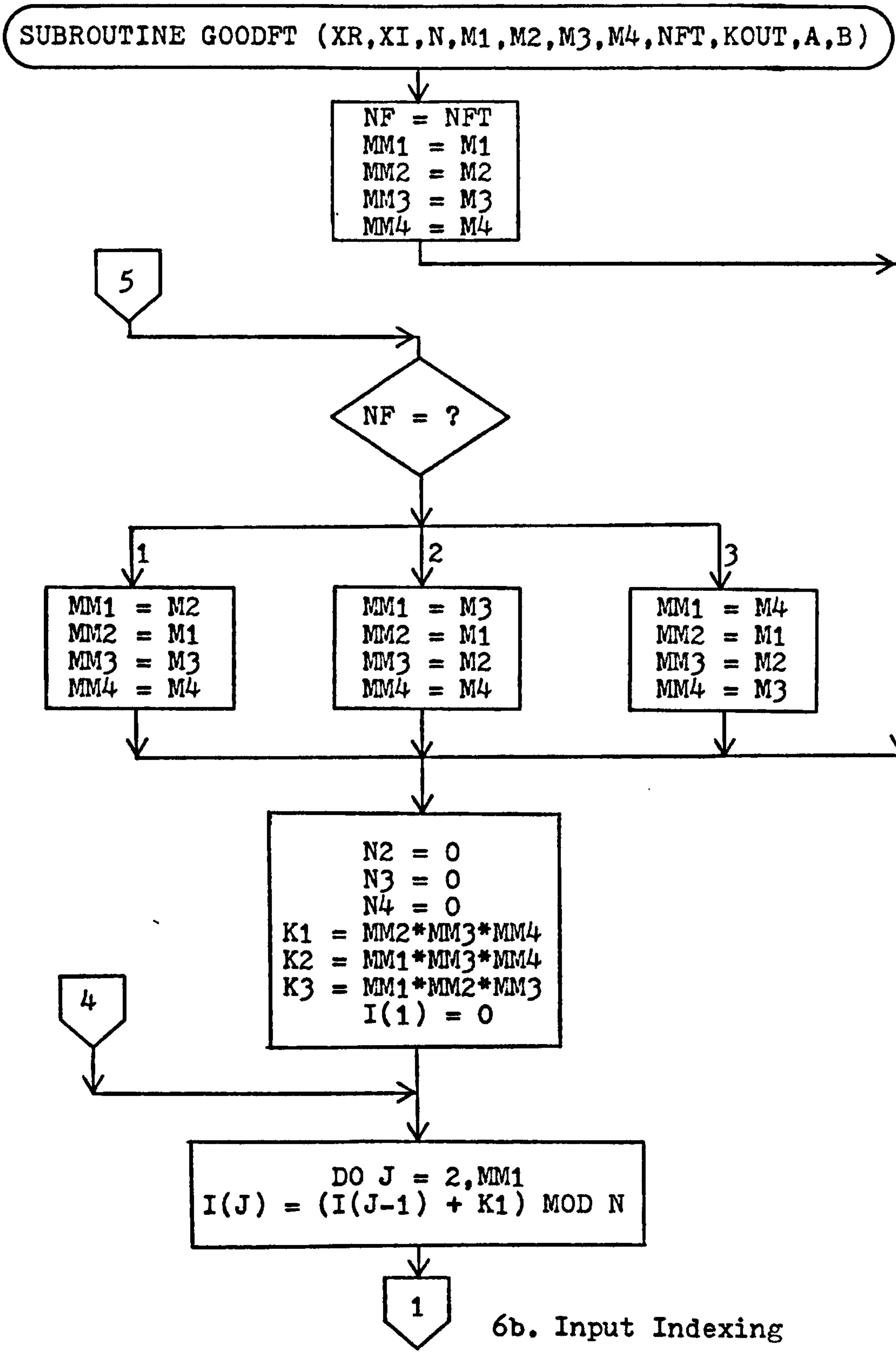
Figure 5. Multiplications and Additions per point



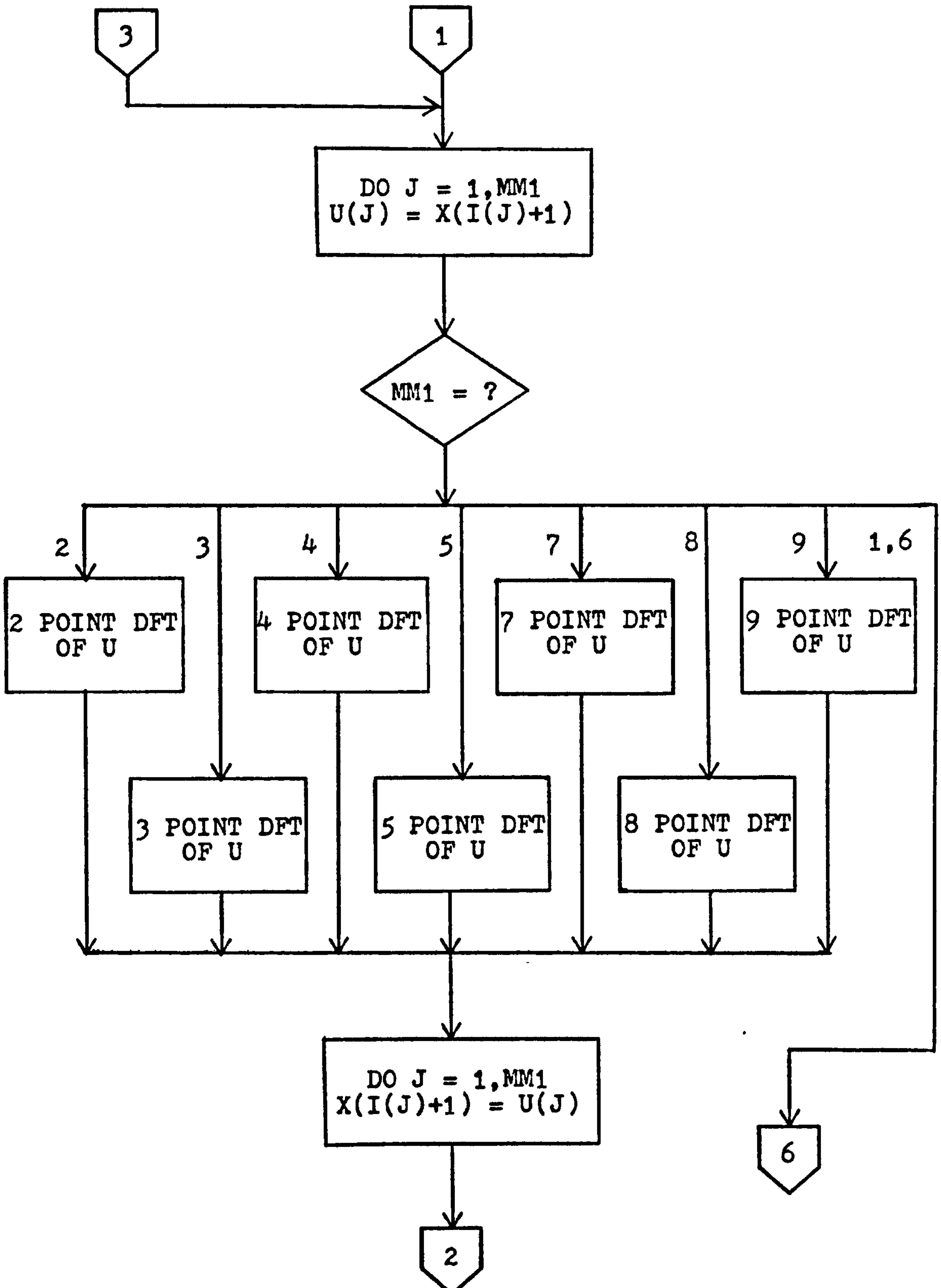
6a. General flowchart of a prime factor FFT

Figure 6. a) General flowchart of a prime factor FFT
b) Input indexing
c) Short transforms
d) More input indexing
e) Output indexing

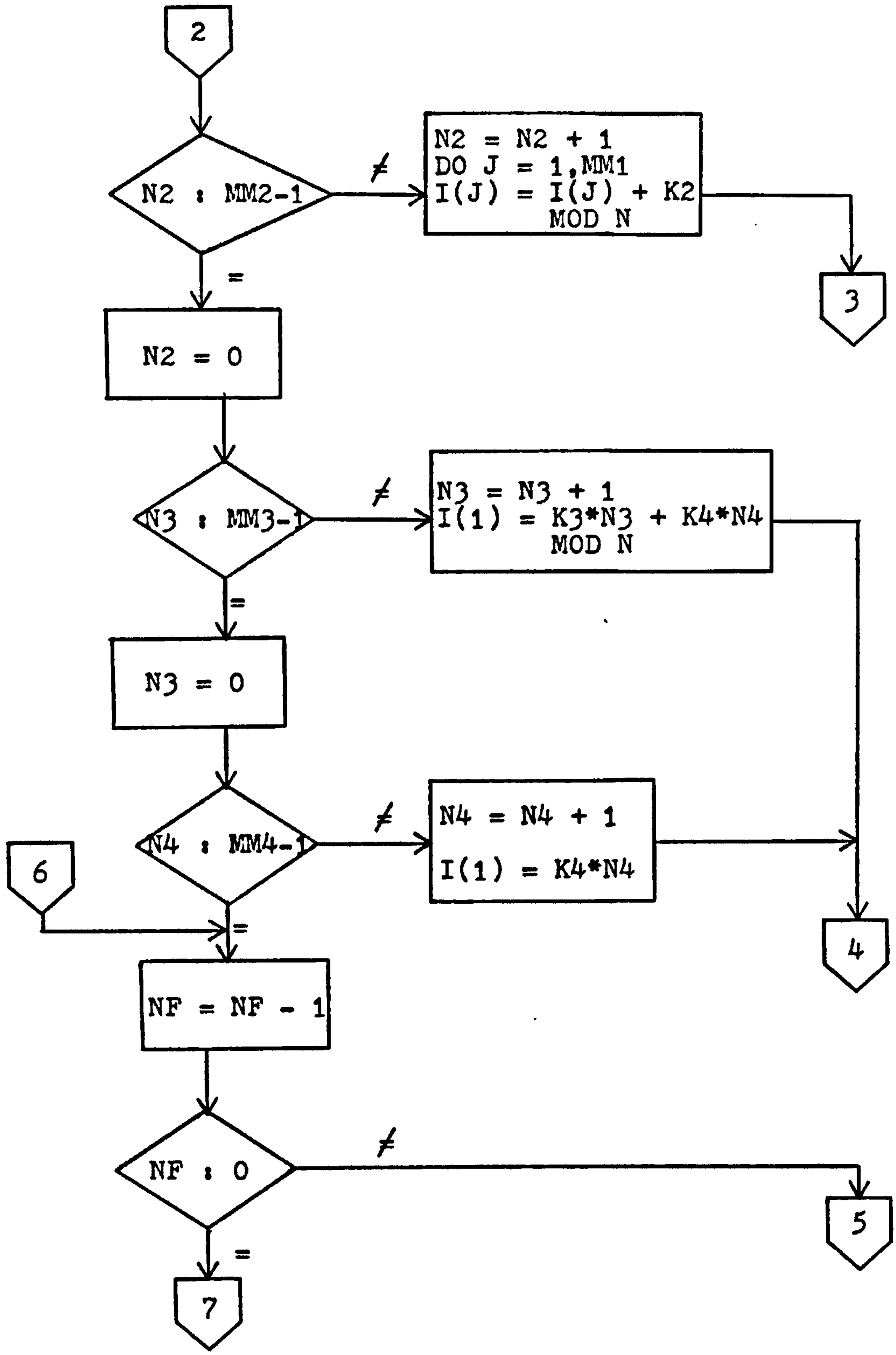
 indicates off page connector in b) through e).



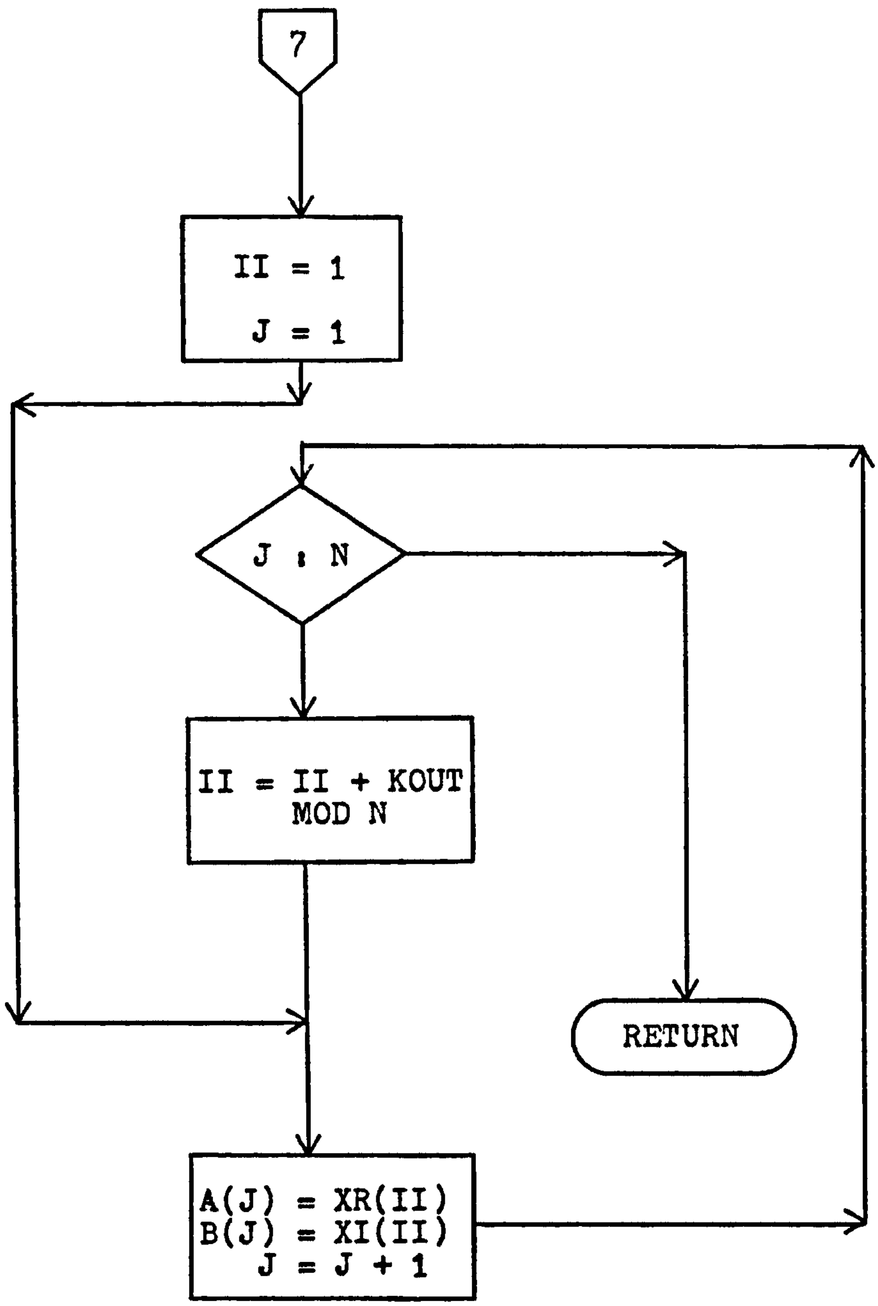
6b. Input Indexing



6c. Short Transforms



6d. More Input Indexing



6e. Output Indexing