1988

The Fast Hartley Transform on a Parallel Processor

Boon Pock Lim

THE FAST HARTLEY TRANSFORM ON A PARALLEL PROCESSOR

BY

LIM, BOON POCK

A thesis submitted in partial fulfillment
of the requirements for the degree
Master of Science
Major in Engineering
South Dakota State University

1988

THE FAST HARTLEY TRANSFORM ON A PARALLEL PROCESSOR

This thesis is approved as a creditable and independent investigation by a candidate for the degree, Master of Science, and is acceptable for meeting the thesis requirements for this degree. Acceptance of this thesis does not imply that the conclusions reached by the candidate are necessarily the conclusions of the major department.

Dr. Douglas B. Hiron — Date
Thesis Adviser

Dr. Virgil G. Ellerbruch — Date
Head, Electrical Engineering Department

ACKNOWLEDGEMENTS

I would like to thank Dr. Douglas B. Miron, Associate Professor of Electrical Engineering, for his guidance and suggestions. Special thanks go to Cheng, Tone Vei for allowing me to use his PANASONIC KX-P1080i printer.

Many have assisted me in my study. I would like to express my deep appreciation to Dr. Ernest L. Buckley, Dean of Engineering, for his encouragement and assistance in obtaining a Lee Foundation scholarship. My sincere appreciation is also extended to the Naito family (Japan), Mr. Lim Chin Lin, Ronnie Chua, the Ang family, and the Koh family (Singapore), for their tangible and/or intangible support.

Finally, I am grateful to my family, especially my two sisters, for their assistance in many ways.

CONTENTS

LIST OF ILLUSTRATIONS

Figures

Tables

CHAPTER ONE

INTRODUCTION

The transform kernel of the Fourier transform is $\exp(-i\omega t)$, i.e. $\cos(\omega t) - i\sin(\omega t)$; where $i=\sqrt{(-1)}$. Obviously the complex arithmetic is needed to obtain the required Fourier transform. In 1942, Hartley[1] proposed a new formulation of the Fourier integral identity by using $\mathrm{cas}(\omega t)$ as the transform kernel, where $\mathrm{cas}(\omega t)=\cos(\omega t) + \sin(\omega t)$, is an abbreviation for cosine and sine. It is called the Hartley transform and has many properties similar to those of the Fourier transform. The important distinctions are that the Hartley transform of a real-valued function is also real valued, and the inverse transformation is the same integral operation as the direct transformation. Furthermore, its evaluation does not involve complex functions. This is a potential advantage if the transform is to be explicitly computed.

The discrete Fourier transform (DFT) has the same transform kernel as the Fourier transform. In 1983, Bracewell[2] introduced the discrete Hartley transform (DHT) by using Hartley's transform kernel, i.e. $\mathrm{cas}(\omega t)$. The DHT can apply to numerical spectral analysis and convolution. Unlike the DFT, no additional program is required for the inverse DHT as it is the same as the direct transformation. If the real and imaginary parts of the DFT are expressly

required then they are directly obtainable as the even and odd parts of the DHT. The power spectrum can also be obtained directly from the DHT without first calculating the real and imaginary parts of the DFT as in the usual way of calculating power spectra.

In 1984, Bracewell[3] worked out a fast algorithm for performing the DHT of a data sequence of N elements in a time proportional to $N\log_2 N$. He proved that the fast Hartley transform (FHT) is as fast as or faster than the fast Fourier transform (FFT) and serves for all uses such as spectral analysis, digital signal processing, and convolution to which the FFT is at present applied. Since then, many discussions on the FHT [5 - 10] and its applications [16], [18] were aroused.

The vector processor (VP) was initiated by Miron in 1985. It is an adjunct to an IBM personal computer in which a sequence of data is passed from the host's memory to a row of mathematical coprocessors, operated on simultaneously by each of a sequence of the coprocessor's instructions, and then the results are passed back to the main memory [21]. It is a single-instruction, multiple-data stream (SIMD) computer system [23]. The purpose of this device is to economically achieve maximum speed in the computer execution of a group of scientific calculations.

Since the FHT is applied on a data sequence of N elements, where N is usually very large, it is a good

candidate to be performed on the vector processor and theoretically, a great improvement on the computer execution time should be obtained.

The Hartley transform, the DHT, and their properties will be presented in the subsequent chapters. The derivation of the DFT from the DHT will also be illustrated. Finally, a software system for computing the DHT on a VP will be discussed and its performance will also be evaluated.

CHAPTER TWO

THE HARTLEY TRANSFORM

2-1 Definition

The old version of the Fourier transform and the inverse Fourier transform were written as

$$S(w) = (2\pi)^{-\frac{1}{2}} \int_{-\infty}^{\infty} V(t) \exp(-iwt)\, dt ,$$

$$V(t) = (2\pi)^{-\frac{1}{2}} \int_{-\infty}^{\infty} S(w) \exp(iwt)\, dw.$$

In order to achieve a symmetrical appearance, Hartley [1] introduced the different pair of formulas

$$\breve{Q}(w) = (2\pi)^{-\frac{1}{2}} \int_{-\infty}^{\infty} V(t) \operatorname{cas}(wt)\, dt ,$$

$$V(t) = (2\pi)^{-\frac{1}{2}} \int_{-\infty}^{\infty} \breve{Q}(w) \operatorname{cas}(wt)\, dw ,$$

where cas is the sum of the cosine and sine, i.e. $cas(x) \equiv \cos(x) + \sin(x)$. However, the new version of the Fourier transform uses $(2\pi)^{1/2}S(w)$ instead of $S(w)$. The result is that the factor $(2\pi)^{-1/2}$ disappears from the transform definition but a factor $(2\pi)^{-1}$ appears in the inversion formula. In terms of frequency, the Fourier transform can be written as

$$F(f) = \int_{-\infty}^{\infty} V(t) \exp(-i2\pi ft)\, dt , \qquad (2\text{-}1)$$

$$V(t) = \int_{-\infty}^{\infty} F(f) \exp(i2\pi ft)\, df. \qquad (2\text{-}2)$$

Adopting this practice leads to

$$H(f) = \int_{-\infty}^{\infty} V(t)\, cas(2\pi ft)\, dt , \qquad (2\text{-}3)$$

$$V(t) = \int_{-\infty}^{\infty} H(f)\, cas(2\pi ft)\, df. \qquad (2\text{-}4)$$

The proof of equation (2-4) can be found in [10]. Therefore, the inverse Hartley transform is indistinguishable from the direct transform.

2-2 Relationship Between H(f) And F(f)

Rewriting equation (2-1) and (2-3), we have

$$F(f) = \int_{-\infty}^{\infty} V(t)\,[\cos(2\pi ft) - i\sin(2\pi ft)]\,dt$$

$$= \int_{-\infty}^{\infty} V(t)\cos(2\pi ft)\,dt -$$

$$i\int_{-\infty}^{\infty} V(t)\sin(2\pi ft)\,dt\,, \qquad (2\text{-}5)$$

and

$$H(f) = \int_{-\infty}^{\infty} V(t)\,[\cos(2\pi ft) + \sin(2\pi ft)]\,dt$$

$$= \int_{-\infty}^{\infty} V(t)\cos(2\pi ft)\,dt +$$

$$\int_{-\infty}^{\infty} V(t)\sin(2\pi ft)\,dt. \qquad (2\text{-}6)$$

Equation (2-5) and (2-6) imply that

$$H(f) = \text{Re } F(f) - \text{Im } F(f). \qquad (2\text{-}7)$$

Let $H(f) = E(f) + O(f)$, where $E(f)$ and $O(f)$ are the even and odd parts of $H(f)$ respectively. Then

$$\text{Re } F(f) = E(f) = \int_{-\infty}^{\infty} V(t) \cos(2\pi ft)\, dt \qquad (2\text{-}8)$$

$$\text{Im } F(f) = -O(f) = -\int_{-\infty}^{\infty} V(t) \sin(2\pi ft)\, dt. \qquad (2\text{-}9)$$

By definition, $E(f) = E(-f)$

and $O(f) = -O(-f)$.

Therefore

$$E(f) = \frac{H(f) + H(-f)}{2}$$

$$= \int_{-\infty}^{\infty} V(t) \cos(2\pi ft)\, dt$$

and

$$O(f) = \frac{H(f) - H(-f)}{2}$$

$$= \int_{-\infty}^{\infty} V(t) \sin(2\pi ft)\, dt.$$

These two integrals are known as the Fourier cosine transform and the Fourier sine transform respectively and have been extensively tabulated[11]. It is true that $E(f)$, $O(f)$, Re $F(f)$, and Im $F(f)$ are real only when $V(t)$ is real.

The discussion above is summarized as follows: The Fourier transform is the even part of the Hartley transform minus i times the odd part; conversely, the Hartley transform is the real part of the Fourier transform minus the imaginary part.

Some good examples can be found in [4].

2-3 Power Spectrum

The power spectrum is well presented in [4].

By definition, the power spectrum $P(f)$ is

$$P(f) = |F(f)|^2$$
$$= [\text{Re } F(f)]^2 + [\text{Im } F(f)]^2.$$

It is also possible to obtain the power spectrum directly from the Hartley transform. Thus

$$P(f) = [\text{Re } F(f)]^2 + [\text{Im } F(f)]^2$$
$$= [E(f)]^2 + [O(f)]^2$$
$$= \frac{[H(f) + H(-f)]^2 + [H(f) - H(-f)]^2}{4}$$
$$= \frac{[H(f)]^2 + [H(-f)]^2}{2}.$$

Thus in lieu of squaring the real and imaginary parts and summing the two values at a given value f, we square and

sum the two values of the Hartley transform at $+f$ and $-f$, and then divide the result by a factor of 2.

2-4 Theorems

Two sorts of theorem are discussed in [4]. The first pertains to operations such as modulation, convolution, and other common operations that may be carried out on a function. This sort of theorem tells what corresponding operation goes on simultaneously in the transform domain. The second kind of theorem deals with relations between functions and their transforms that can typically be expressed in the form of an equation. They are reproduced from [4] and listed in Table 2.1 and Table 2.2 respectively. A worthy point to note is the convolution theorem. If one or both of the functions entering into the convolution are even, then the Hartley theorem is the same as the Fourier theorem, i.e. $H_1(f)\ H_2(f)$.

Table 2.1 Theorems for the Fourier and Hartley transforms

Theorem	$V(t)$	$F(f)$	$H(f)$
Similarity	$V(t/T)$	$\lvert T\rvert F(Tf)$	$\lvert T\rvert H(Tf)$
Addition	$V_1(t)+V_2(t)$	$F_1(f)+F_2(f)$	$H_1(f)+H_2(f)$
Reversal	$V(-t)$	$F(-f)$	$H(-f)$
Shift	$V(t-T)$	$e^{-i2\pi Tf}F(f)$	$\cos(2\pi Tf)H(f)$ $+\sin(2\pi Tf)H(-f)$
Modulation	$V(t)\cos 2\pi f_0 t$	$\frac{1}{2}[F(f-f_0)+$ $F(f+f_0)]$	$\frac{1}{2}[H(f-f_0)+$ $H(f+f_0)]$
Convolution	$V_1(t)*V_2(t)$	$F_1(f)F_2(f)$	$\frac{1}{2}[H_1(f)H_2(f)-$ $H_1(-f)H_2(-f)+$ $H_1(f)H_2(-f)+$ $H_1(-f)H_2(f)]$
Product	$V_1(t)V_2(t)$	$F_1(f)*F_2(f)$	$\frac{1}{2}[H_1(f)*H_2(f)-$ $H_1(-f)*H_2(-f)+$ $H_1(f)*H_2(-f)+$ $H_1(-f)*H_2(f)]$
Autocorrelation	$V(t)@V(t)$	$\lvert F(f)\rvert^2$	$\frac{1}{2}\{[H(f)]^2+[H(-f)]^2\}$
Derivative	$V'(t)$	$i2\pi fF(f)$	$-2\pi fH(-f)$
2nd derivative	$V''(t)$	$-4\pi^2 f^2F(f)$	$-4\pi^2 f^2H(f)$

Table 2.2 Theorems for relations between domains

Theorem	Property	Fourier relation	Hartley relation
Infinite integral	$\int_{-\infty}^{\infty} V(t)\,dt$	$= F(0)$	$= H(0)$
First moment	$\int_{-\infty}^{\infty} tV(t)\,dt$	$= F'(0)/(-i2\pi)$	$= -H'(0)/2\pi$
Second moment	$\int_{-\infty}^{\infty} t^2V(t)\,dt$	$= -F''(0)/4\pi^2$	$= -H''(0)/4\pi^2$
Centroid	$\dfrac{\int_{-\infty}^{\infty} tV(t)\,dt}{\int_{-\infty}^{\infty} V(t)\,dt}$	$= iF'(0)/(2\pi F(0))$	$= -H'(0)/(2\pi H(0))$

CHAPTER THREE

THE DISCRETE HARTLEY TRANSFORM

3-1 Definition

The discrete Fourier transform (DFT) and its inverse have the standard form

$$F(k) = N^{-1} \sum_{n=0}^{N-1} f(n)\exp(-i2\pi nk/N) \qquad (3\text{-}1)$$

$$f(n) = \sum_{k=0}^{N-1} F(k)\exp(i2\pi nk/N) \qquad (3\text{-}2)$$

The function $f(n)$ may be the discrete representation of an underlying continuous waveform or may be a function of a variable that is basically discrete.

Bracewell[2] defined the discrete Hartley transform (DHT) of a finite length sequence and its inverse as

$$H(k) = N^{-1} \sum_{n=0}^{N-1} f(n)\mathrm{cas}(2\pi nk/N) \quad , \qquad (3\text{-}3)$$

$$f(n) = \sum_{k=0}^{N-1} H(k)\mathrm{cas}(2\pi nk/N) \quad , \qquad (3\text{-}4)$$

$$0 \leq n \leq (N-1) \quad ,$$

where $\mathrm{cas}(x) \equiv \cos(x) + \sin(x)$. The proof of equation (3-4) can be found in [4], pp. 28. The DHT is real if $f(n)$ is real. Comparing these equations, one can see that the DHT and the DFT are closely related.

3-2 Relationship Between The DHT And The DFT

Rewriting equation (3-1) and (3-3), we have

$$F(k) = N^{-1}\sum_{n=0}^{N-1} f(n)[\cos(2\pi nk/N) - i\sin(2\pi nk/N)]$$

$$= N^{-1}\sum_{n=0}^{N-1} f(n)\cos(2\pi nk/N) -$$

$$iN^{-1}\sum_{n=0}^{N-1} f(n)\sin(2\pi nk/N)$$

and

$$H(k) = N^{-1}\sum_{n=0}^{N-1} f(n)[\cos(2\pi nk/N) + \sin(2\pi nk/N)]$$

$$= N^{-1}\sum_{n=0}^{N-1} f(n)\cos(2\pi nk/N) +$$

$$N^{-1}\sum_{n=0}^{N-1} f(n)\sin(2\pi nk/N).$$

As in the continuous case, the DHT possesses even and odd parts

$$H(k) = E(k) + O(k)$$

where $E(k) = N^{-1}\sum_{n=0}^{N-1} f(n)\cos(2\pi nk/N)$,

$$O(k) = N^{-1}\sum_{n=0}^{N-1} f(n)\sin(2\pi nk/N) \quad .$$

If k is replaced by (N - k),

$$E(N-k) = N^{-1}\sum_{n=0}^{N-1} f(n)\cos(2\pi n - 2\pi nk/N)$$

$$= N^{-1}\sum_{n=0}^{N-1} f(n)\cos(2\pi nk/N)$$

$$= E(k)$$

and

$$O(N-k) = N^{-1}\sum_{n=0}^{N-1} f(n)\sin(2\pi n - 2\pi nk/N)$$

$$= -N^{-1}\sum_{n=0}^{N-1} f(n)\sin(2\pi nk/N)$$

$$= -O(k) \ .$$

Thus

$$E(k) = \frac{[H(k) + H(N-k)]}{2}$$

and

$$O(k) = \frac{[H(k) - H(N-k)]}{2} \ .$$

In the case of k = 0 and k = N/2, one can prove that

$$E(0) = H(0) \quad , \quad O(0) = 0$$

$$E(N/2) = H(N/2) \quad , \quad O(N/2) = 0.$$

From the definition of the DFT, it is apparent that $F(k)$ can be formed from the DHT's even and odd parts by

$$F(k) = E(k) - jO(k)$$

$$= \frac{[H(k) + H(N-k)]}{2} - j\frac{[H(k) - H(N-k)]}{2}$$

$$= \frac{[H(N-k) + H(k)]}{2} + j\frac{[H(N-k) - H(k)]}{2} \ .$$

Conversely, to form $H(K)$ when $F(K)$ is available

$$H(K) = \text{Re } F(K) - \text{Im } F(K) .$$

These relations are strictly analogous to those obtained previously for the continuous variable.

A number of examples that illustrates the charateristics of the DHT can also be found in [4].

3-3 Theorems

The theorems of the DHT discussed here are based upon [4]. Just as there is a Hartley transform theorem for every theorem that applies to the Fourier transform, there are also corresponding theorems for the discrete transforms. The theorems such as convolution, autocorrelation, and first value, etc. are listed in Table 3.1 and Table 3.2.

The proofs and discussions on some theorems such as reversal, addition, shift, product, and convolution are well presented in [4].

One may note that the mean value of the sequence $f(n)$ is given by $H(0)$ and that the mean square value of $f(n)$ is given by ΣH^2.

Some of the theorems for the two different transforms correspond exactly, as is the case with $\Sigma H(K) = f(0)$ and $\Sigma f(n) = N \times H(0)$, but some exhibit differences.

The only theorem not listed in the table is the

stretch or similarity theorem. It states that if a sequence $f(n)$ is stretched to double its length by inserting a zero element after each given element, then the elements of the original DHT are repeated. For example,

{1 2 3 4} has the DHT {2.5 -1 -0.5 0} ,

then {1 0 2 0 3 0 4 0} has the DHT

{2.5 -1 -0.5 0 2.5 -1 -0.5 0}.

The properties of the DHT commend themselves for application to numerical analysis. The fact that the transform values are real is a convenience in managing calculations. In addition, the reversibility of the transform is helpful as one does not need to keep track of which domain one is in. Furthermore, several of the theorems for the Fourier transform have different forms according to the domain, a concern that is obviated with the DHT. The factor N is domain-dependent and could be a normalization factor or calibration factor to be applied at the end of a numerical calculation. Experience shows that the last step is the place to consolidate proportionality factors and so the departure from strict reversibility represented by the factor N is not computationally important [4].

Table 3.1 Theorems for operations on discrete transforms

Theorem	Function $f(n)$	DFT $F(k)$	DHT $H(k)$
Reversal	$f(-n)$	$F(-k)$	$H(-k)$
Addition	$f_1(n)+f_2(n)$	$F_1(k)+F_2(k)$	$H_1(k)+H_2(k)$
Shift	$f(n-T)$	$e^{-i2\pi Tk/N}F(k)$	$\cos(2\pi Tk/N)H(k)$ $+\sin(2\pi Tk/N)H(N-k)$
Convolution	$f_1(n)*f_2(n)$	$NF_1(k)F_2(k)$	$\frac{1}{2}N[H_1(k)H_2(k)-$ $H_1(-k)H_2(-k)+$ $H_1(k)H_2(-k)+$ $H_1(-k)H_2(k)]$
Product	$f_1(n)f_2(n)$	$F_1(k)*F_2(k)$	$\frac{1}{2}N[H_1(k)*H_2(k)-$ $H_1(-k)*H_2(-k)+$ $H_1(k)*H_2(-k)+$ $H_1(-k)*H_2(k)]$
Autocorrelation	$f(n)\circledast f(n)$	$\frac{1}{2}N\lvert F(k)\rvert^2$	$N\{[H(k)]^2+[H(-k)]^2\}$
Derivative	$f'(n)$	$i2\pi kF(k)$	$2\pi kH(-k)$
2nd derivative	$f''(n)$	$-4\pi^2k^2F(k)$	$-4\pi^2k^2H(k)$

Table 3.2 Theorems on relations for discrete transforms

Sum of sequence	$\sum_{n=0}^{N-1} f(n) = NF(0) = NH(0)$
First value	$f(0) = \sum_{k=0}^{N-1} F(k) = \sum_{k=0}^{N-1} H(k)$
Quadratic content	$\sum_{n=0}^{N-1} [f(n)]^2 = N \sum_{k=0}^{N-1} \|F(k)\|^2 = N \sum_{k=0}^{N-1} \|H(k)\|^2$

CHAPTER FOUR

THE FAST HARTLEY TRANSFORM

Since the discrete Hartley transform (DHT) is closely related to the discrete Fourier transform (DFT), one can predict that the fast Fourier transform (FFT) algorithm can be converted to a fast Hartley transform (FHT) with some modifications. The FHT can be expected to transform one real array of length N in half the time that it takes the FFT to process a complex array of length N. Buneman [10] presented the similarities between DHT and DFT and showed that the role played by the imaginary part in the complex Fourier transform is taken on by the real Hartley transform recorded backwards so that one can convert an FFT program into an FHT program with only a few indexing changes.

About one year after his proposal of the DHT, Bracewell [3] published the first fast algorithm for performing the DHT which was basically the same as Cooley and Tukey's method [12] for the FFT, or so-called decimation-in-time (DIT) algorithm (It is also called a decimation-in-time radix-2 algorithm or just a radix-2 algorithm.) Since then, the various methods for computation of the FFT were examined and attempted for the FHT. The methods such as decimation-in-frequency (DIF) FHT [5], split-radix (SR) FHT [9], in-place FHT [6], and radix-4 FHT [8] had been worked out for computing the DHT. The total numbers of the operations of

the first two methods and the DIT are summarized as follows [9]:

	Additions	Multiplications
DIT	$2N(\log_2 N-1)+2$	$N(\log_2 N-2)+2$
DIF	$3/2\ N(\log_2 N-1)+2$	$N(\log_2 N-3)+4$
SR	$(2N-4)(\log_4 N)$	$(N-4)(\log_4 N)$

For ease of comparison, various values of N are calculated and shown in Table 4.1 and Table 4.2.

Table 4.1 Number of real multiplications to compute an N-point DHT

N	DIT	DIF	SR
16	34	20	24
32	98	68	84
64	258	196	180
128	642	516	496
256	1538	1284	1008
512	3586	3076	2540
1024	8194	7172	5100

Table 4.2 Number of real additions to compute an N-point DHT

N	DIT	DIF	SR
16	98	74	56
32	258	194	180
64	642	482	372
128	1538	1152	1008
256	3586	2690	2032
512	8194	6146	5100
1024	18434	13826	10220

A comparison of operation counts on various methods of the FHT can be found in [7], [8], and [9]. The application of the same method but with different approach will cause a minor difference in operation counts. Although [7] and [8] concluded that the FHT algorithms do not give any increase in performance over the existing fastest real-valued FFT algorithms such as Winograd's FFT algorithm [14] and Bergland and Dolan's algorithm [15], one may predict that the FHT has a great potential in some situations as its real-valued function nature and the equivalence of the forward and inverse DHT may justify the cost. Hou[13] presented his algorithm and showed that in lower order transforms the number of nontrivial, real arithmetic operations in his FHT is about the same as in the Winograd's FFT.

One may note that N^{-1} in equation (3-3) usually is regarded as the normalization factor in the FHT algorithm and is neglected until the very end.

Equation (3-3) can be written in matrix form as

$$[H] = N^{-1} [CAS] [X]. \qquad (4-1)$$

Where [H] and [X] are N x 1 matrices, N^{-1} is a scalar, and

$$
\begin{array}{rcl}
 & \begin{matrix} k \\ \| \\ 0 \\ 1 \\ 2 \\ : \\ : \\ N-1 \end{matrix} & \begin{matrix} n = 0, 1, 2, \ldots\ldots, N-1 \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \\
[CAS] = & \left[\begin{matrix} \\ \\ \\ \cos(2\pi nk/N) + \sin(2\pi nk/N) \\ \\ \\ \\ \\ \end{matrix} \right] &
\end{array}
$$

For a computer language which has powerful matrix manipulations, e.g. APL, equation (4-1) is very useful and should be considered.

If the cas function is carefully analyzed, one may notice that

when n is replaced by (N - n)

$$\cos[2\pi(N-n)k/N] = \cos[2\pi nk/N] \ ;$$

$$\sin[2\pi(N-n)k/N] = -\sin[2\pi nk/N] \ ,$$

and when k is replaced (N - k)

$$\cos[2\pi n(N-k)/N] = \cos[2\pi nk/N] \ ;$$

$$\sin[2\pi n(N-k)/N] = -\sin[2\pi nk/N] \ .$$

Also, when n = 0 or k = 0, the values of the cosine terms are all '1's and the sine terms are all '0's.

When n = N/2 or k = N/2, the values of the sine terms are also all '0's but the values of the cosine terms are varied, that is

$$\cos(\pi k) = (-1)^k \qquad \text{and}$$

$$\cos(\pi n) = (-1)^n.$$

These imply that

$$[COS] = \begin{array}{c} \\ 0 \\ \\ 2 \\ \vdots \\ \\ N/2 \\ \vdots \\ \\ N-1 \end{array} \begin{array}{c} \begin{array}{cccccccc} 0 & 1 & \cdots & & N/2 & \cdots & & N-1 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & \cdots & 1 & 1 \\ 1 & Z_1 & Z_2 & \cdots & -1 & \cdots & Z_2 & Z_1 \\ 1 & Z_3 & Z_4 & \cdots & 1 & \cdots & Z_4 & Z_3 \\ \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots \\ 1 & -1 & 1 & -1 & 1 & \cdots & 1 & -1 \\ \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots \\ 1 & Z_3 & Z_4 & \cdots & -1 & \cdots & Z_4 & Z_3 \\ 1 & Z_1 & Z_2 & \cdots & 1 & \cdots & Z_2 & Z_1 \end{bmatrix} \end{array}$$

and

$$[SIN] = \begin{array}{c} \\ 0 \\ \\ 2 \\ \vdots \\ \\ N/2 \\ \vdots \\ \\ N-1 \end{array} \begin{array}{c} \begin{array}{cccccccc} 0 & \cdots & & N/2 & \cdots & & N-1 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & Y_1 & Y_2 & \cdots & 0 & \cdots & -Y_2 & -Y_1 \\ 0 & Y_3 & Y_4 & \cdots & 0 & \cdots & -Y_4 & -Y_3 \\ \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots \\ 0 & -Y_3 & -Y_4 & \cdots & 0 & \cdots & Y_4 & Y_3 \\ 0 & -Y_1 & -Y_2 & \cdots & 0 & \cdots & Y_2 & Y_1 \end{bmatrix} \end{array}$$

Recall that

$$[CAS] = [COS] + [SIN],$$

which means a quarter of each sine and cosine matrix shall be calculated so that the complete matrix can be formed via move and negate instructions. Since the time

taken for calculating the trigonometric function is much longer than moving the data to the various locations, a time reduction should be achieved by using this method although it increases the complexity of the program.

Since $cas(x) = \cos(x) + \sin(x)$,

$\cos(\pi/4) = \sin(\pi/4) = 2^{-\frac{1}{2}}$,

it may be faster to replace $cas(x)$ by $2^{\frac{1}{2}}\sin[x+(\pi/4)]$ or $2^{\frac{1}{2}}\cos[x-(\pi/4)]$ on some computers.

CHAPTER FIVE

THE VECTOR PROCESSOR

The vector processor (VP) was initiated by Miron in 1985. It is an adjunct to a personal computer in which a sequence of data is passed from the host's memory to a row of mathematical coprocessors (MCs), operated on simultaneously by each of a sequence of coprocessor instructions, and then the results are passed back to main memory. It is a single-instruction, multiple-data (SIMD) auxiliary computer. Its purpose is to speed up all those operations which can be expressed in vector-matrix forms by parallel execution of the data operations [21].

The general arrangement of the major elements of the VP is shown in figure 5.1. The vector control unit (VCU) in the vector processor is as important as the general purpose processor (GPP) in the personal computer. It consists of a vector instructions decoder, a sequential load/store control unit, and a parallel execution control unit. Its function is to co-ordinate the operation between the GPP and the row of MCs. If the vector operation (parallel mode) is not required, the VCU will connect one of the MCs to the GPP as in the conventional architecture of figure 5.2 so that faster speed in executing ordinary arithmetic operations can be achieved. When a vector instruction appears, it is decoded by the VCU and appropriate action taken. The effect of these

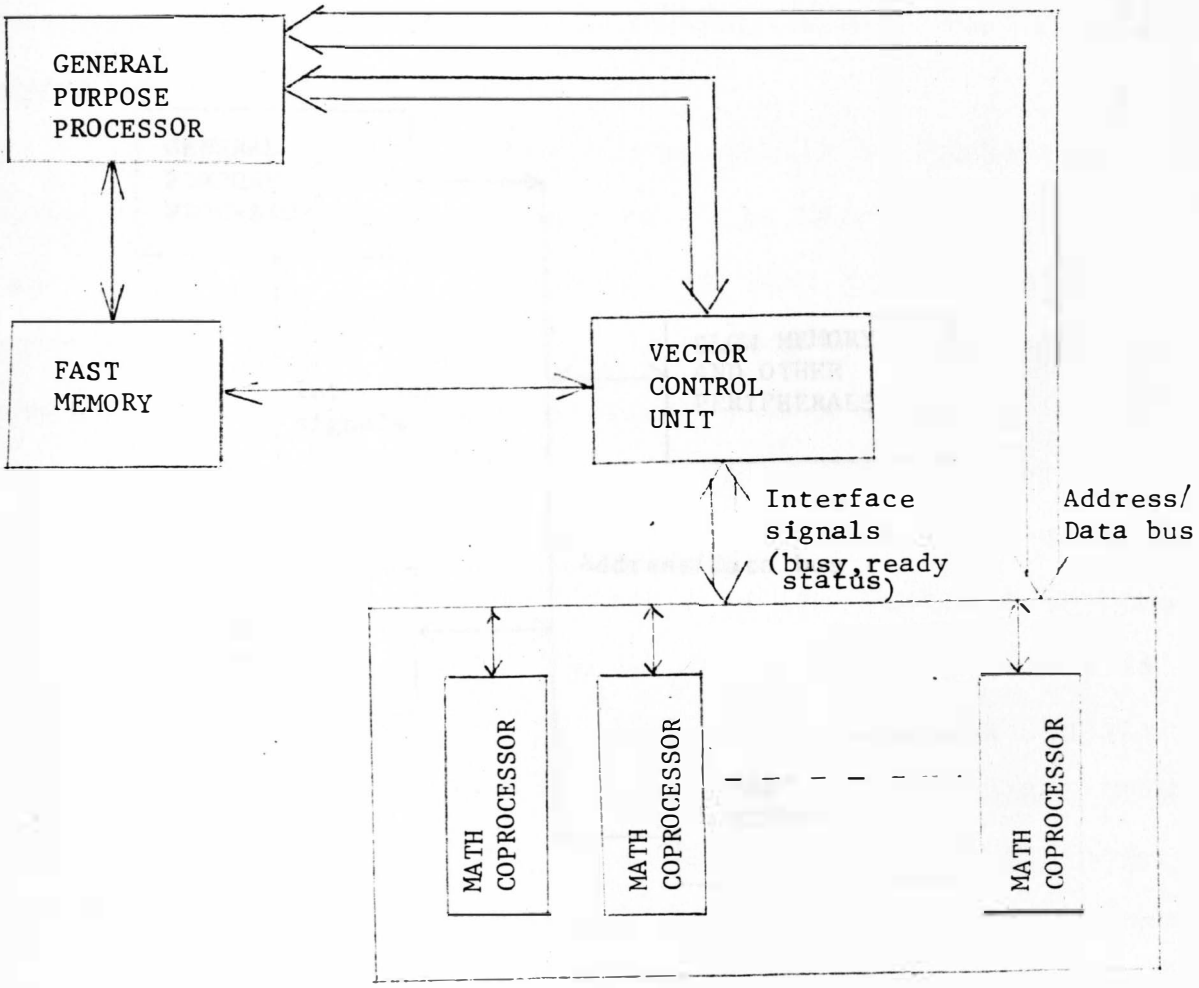


Figure 5.1 Vector Processor Architecture

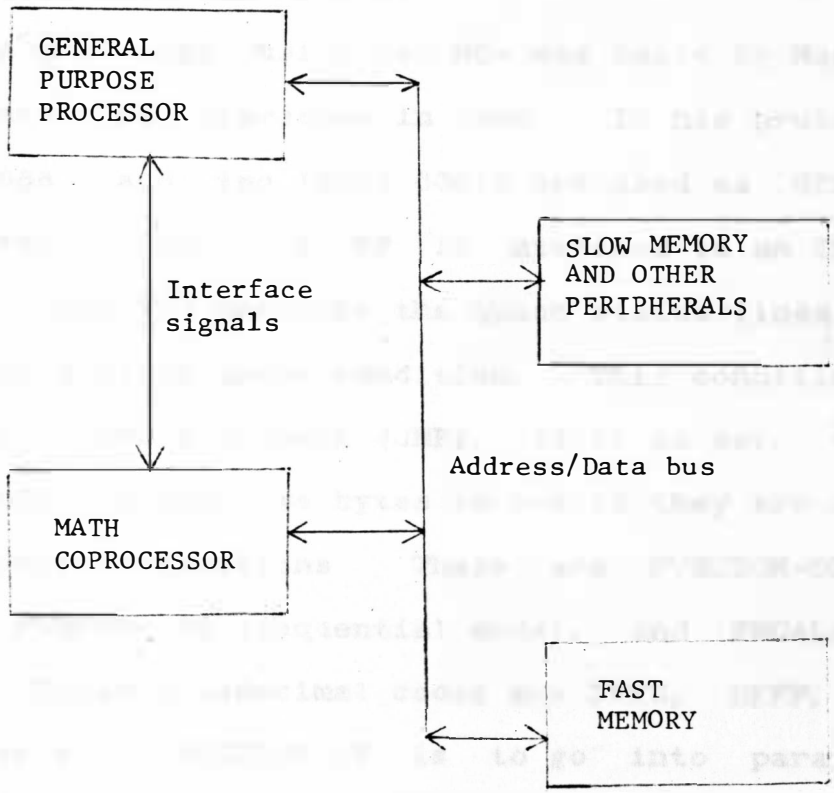


Figure 5.2 A Conventional Microcomputer Architecture

actions is to load the elements of the required vectors sequentially from fast memory into the MCs, do the required operations simultaneously, and then store the results sequentially into fast memory.

A prototype using two MCs was built by Manja [22] to demonstrate the operation in 1986. In his prototype, an Intel 8088 and two Intel 8087s are used as GPP and MCs respectively, and the VP is attached to an IBM personal computer. The VCU monitors the queue status lines of the GPP to detect a clear-queue condition. This condition will be set after a jump statement (JMP). If it is set, the decoder will decode the next two bytes to see if they are one of the three vector instructions. These are FVECTOR-OP (parallel mode), FVECTOR-SQ (sequential mode), and FSCALAR (scalar mode). Their hexadecimal codes are DFFD, DFFF, and DFFE respectively. FVECTOR-OP is to go into parallel mode. FVECTOR-SQ is used for loading/storing the data to/from the MCs, whereas FSCALAR is for returning to the conventional mode and also acting as a reset between the other two modes. In each mode, the VCU selects the MC by its READY and queue status lines. The functional blocks and signals managed in the VCU are shown in figure 5. 3.

As mentioned earier the purpose of the VP is to speed up the processing time of the data. A measure of the improvement to be expected is defined as the ratio of the total time to process the data with a single MC to the time

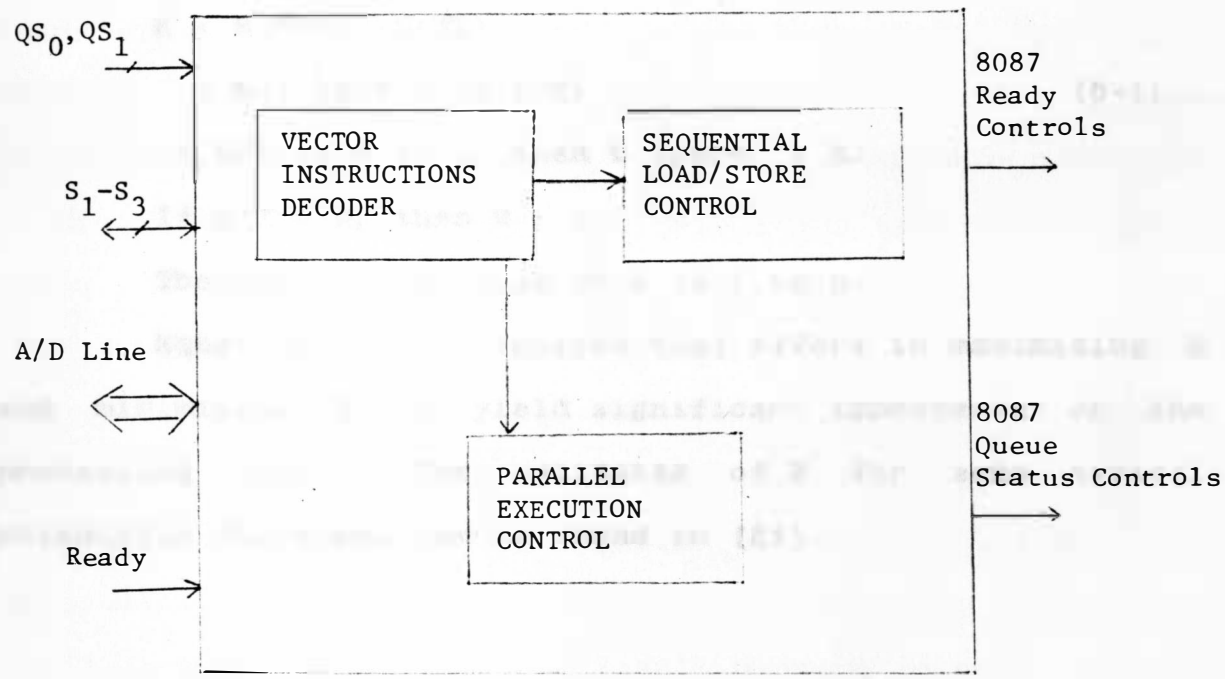


Figure 5.3 Vector Controller

with M MCs in a VP [21]. For a given function, let

E = execution time for one result element,

T = total data transfer time for that result element,

M = the number of MCs,

and R = improvement ratio.

By definition,

$$R = M(E+T)/(E+MT)$$

$$= M(1+(E/T))/(M+E/T) \qquad (5\text{-}1)$$

If E/T >> M >> 1, then R appro. = M.

If E/T = 0, then R = 1.

Therefore, the range of R is 1 to M.

Equation (5-1) implies that effort in maximizing E and minimizing T will yield significant improvement on the processing time. The estimates of R for some typical scientific functions can be found in [21].

CHAPTER SIX

THE FHT ON A VECTOR PROCESSOR

6-1 Selection of an Algorithm

A program is written for computing the discrete Hartley transform (DHT) on a vector processor (VP). Some fast Hartley transform (FHT) algorithms that have been published are for a single processor [3 - 10]. These algorithms can be implemented on the VP with some modifications. An algorithm with less processing time for a single processor may not necessarily be the best algorithm for the VP. This is due to the VP's configuration. An Intel 8088 and some Intel 8087s are assumed to be the general purpose processor (GPP) and the mathematical coprocessors (MCs) respectively for a VP. As proved in chapter five, the greatest improvement on the processing time will be achieved if the execution time for one result element is much greater than the total data transfer time for that result element. This means the data set involved in parallel processing should be arranged in such a way that a result can be derived from the original data set and not the results previously computed from the data set. This is called data independence. If the elements of the data set are not independent of each other, parallel processing cannot be applied. Furthermore, comparison of the intermediate result must be avoided. Comparison can only be done in scalar or sequential

mode. It will defeat the purpose of parallel processing. The Intel 8087 has eight internal registers [24], [25]. Fully using these registers to keep the intermediate results will definitely reduce the data transfer time. Although sometimes it is done at the expense of the memory storage, data independence, avoiding comparison, and reducing the data transfer time are the key considerations for programming the parallel processing routine on a VP.

The result from the above discussion is the use of the algorithm of Bold [7] for computing the DHT on a VP. A detailed discussion on the algorithm will be presented in the next section.

6-2 The FHT Algorithm

Consider a data sequence

$f(n) = \{a_1\ a_2\ b_1\ b_2\ c_1\ c_2\ d_1\ d_2\}$,

we can rewrite it as

$f(n) = g_1(n) + g_2(n-1)$,

where $g_1(n) = \{a_1\ 0\ b_1\ 0\ c_1\ 0\ d_1\ 0\}$,

$g_2(n) = \{a_2\ 0\ b_2\ 0\ c_2\ 0\ d_2\ 0\}$,

and $g_2(n-1) = \{0\ a_2\ 0\ b_2\ 0\ c_2\ 0\ d_2\}$.

Suppose $\{a_1\ b_1\ c_1\ d_1\}$ has the DHT

$\{\alpha_1\ \beta_1\ \tau_1\ \partial_1\}$

and $\{a_2\ b_2\ c_2\ d_2\}$ has the DHT

$\{\alpha_2\ \beta_2\ \tau_2\ \partial_2\}$.

By applying the stretch theorem and the shift theorem for the DHT, we can show that

$$G_1(K) = \{\alpha_1\ \beta_1\ \tau_1\ \partial_1\ \alpha_1\ \beta_1\ \tau_1\ \partial_1\},$$

$$G_2(K) = \{\alpha_2\ \beta_2\ \tau_2\ \partial_2\ \alpha_2\ \beta_2\ \tau_2\ \partial_2\},$$

and $G_2(K-1) = \cos(2\pi K/N)G_2(K) + \sin(2\pi K/N)G_2(N-K)$.

Therefore, the DHT of a sequence $f(n)$ can be written as

$$G(K) = G_1(K) + \cos(2\pi K/N)G_2(K) + \sin(2\pi K/N)G_2(N-K).$$

Adopting this approach, Bracewell [4] shows that the general decomposition formula that produces the discrete Hartley transform (DHT) for a sequence $f(n)$ of N elements, where $N = 2^P$, is

$$H(K) = H_{odd}(K) + H_{even}(K)\cos(2\pi K/N) + H_{even}(N-K)\sin(2\pi K/N), \qquad (6\text{-}1)$$

where $k = 0, 1, 2, 3, \ldots\ldots, (N-1)$, $H_{odd}(K)$ and $H_{even}(K)$ are the DHT of the odd and the even numbered terms in the sequence f(n) respectively.

Since $\cos(2\pi((N/2)+K)/N) = -\cos(2\pi K/N)$,

$\sin(2\pi((N/2)+K)/N) = -\sin(2\pi K/N)$,

and the terms in H_{odd} and H_{even} repeat modulo N/2 [4], equation (6-1) can be rewritten as

$$H(K) = H_{odd}(K) + H_{even}(K)\cos(2\pi K/N) + H_{even}(N-K)\sin(2\pi K/N) \qquad (6\text{-}2)$$

and $$H((N/2)+K) = H_{odd}(K) - H_{even}(K)\cos(2\pi K/N) - H_{even}(N-K)\sin(2\pi K/N), \qquad (6\text{-}3)$$

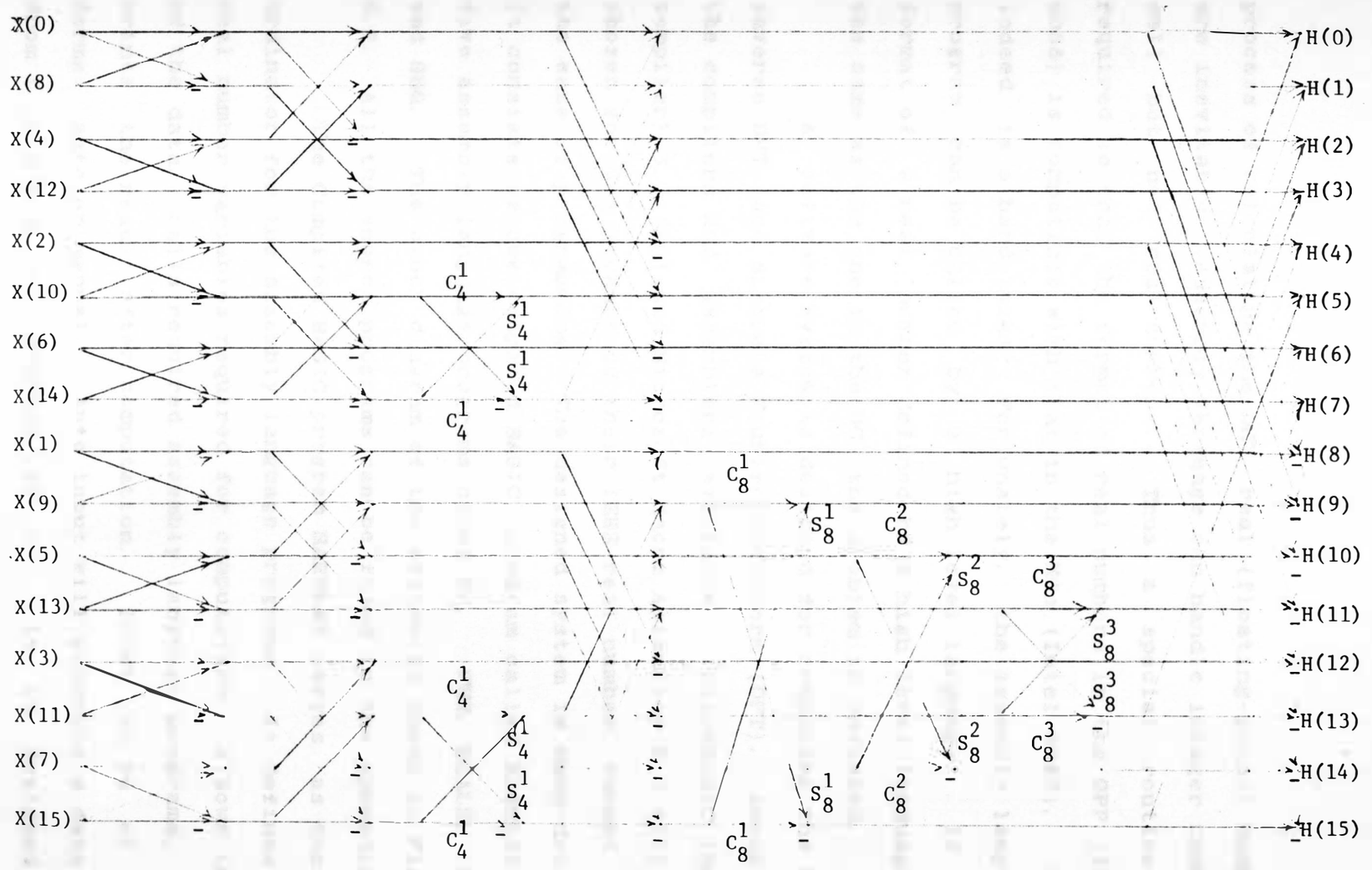
where k = 0, 1, 2, 3, ... , ((N/2)-1).

Since $H_{odd}(k)$ and $H_{even}(k)$ can be derived by continued decomposition until two-element sequences are reached, the complete breakdown is expressible in terms of the original data. This is similar to the form used by Cooley and Tukey [12] for computing the DFT. Consequently, a bit-reversing operation is required before the computation begins. The basic algorithm for a bit-reversing operation can be found in [26]. The signal flow graph as developed from equation (6-2) and equation (6-3) for a 16-element sequence is shown in figure 6.1.

6-3 System Design

The algorithm used for computing the FHT on a VP is based on the work of Bold [7]. However, some modifications have been made so that optimization of processing time can be achieved. The algorithm for the bit-reversing operation in Bold's work involves comparison of the data. Hence it is not suitable for programming on the VP and is replaced by the work done by Brigham [26]. Because the VP has its own three unique instructions, i.e. parallel mode, sequential mode, and scalar mode, which cannot be recognized by the high level computer languages (such as FORTRAN, BASIC, PASCAL, and C), assembly language must be used. By using assembly language, the VP's three instructions can be coded as data constants, using their corresponding hexadecimal numbers. In the

Figure 6.1 FHT signal flow graph for N = 16



$S_N^n = \sin((2\pi/N)n)$, $C_N^n = \cos((2\pi/N)n)$

process of calculating the DHT, real (floating-point) numbers are inevitable. Assembly language can handle integer numbers well but not real numbers. Thus a special routine is required so that the format of real numbers in the GPP (Intel 8088) is compatible with that in the MCs (Intel 8087). This indeed is a hard task! Fortunately, the assembly language program can be called by a high level language. If the format of a real number defined by a high level language is the same as the one in the MC, the problem is obviated.

A software system is designed for computing the DHT, inverse DHT, and discrete Fourier transform (DFT). Among all the compilers and assemblers available, QuickBASIC (BASIC compiler) 3.0 [28] and Microsoft Macro Assembler 5.0 [27] are chosen for use because of their IEEE real number format and the ease of programming. The designed system is menu-driven. It consists of one compiled BASIC program called **HARTLEY** and five assembly language programs named P2, **FHT**, **SCALE**, **DFT**, and **SEQ**. The block diagram of the system is shown in Figure 6.2. All the source programs can be found in the appendix.

The Compiled BASIC program **HARTLEY** serves as the co-ordinator for the assembly language programs. It defines the real number variables required for computation, allows input of the data, calls required assembly language programs, and prints the result after computation. Input can be of two forms: auto or manual. Auto input will generate a data set from 1 to N with an interval of 1. It is designed for

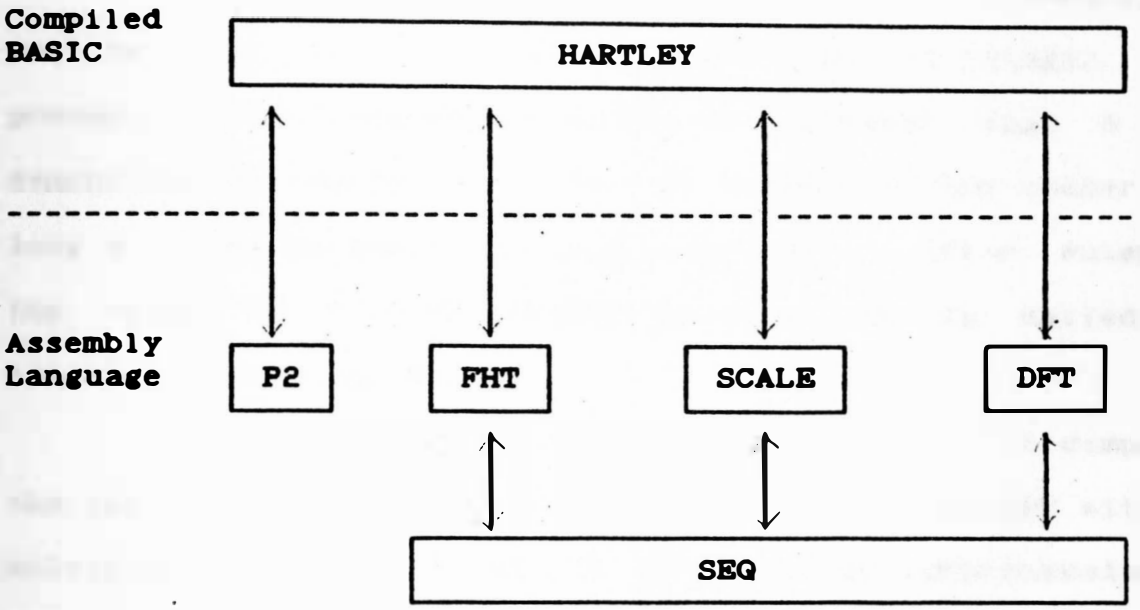


Figure 6.2 The DHT computation system

testing and demonstration. After computing the DHT, the result can be shown on the terminal or sent to the printer.

As mentioned before, for a sequence of N elements, N must be equal to 2^P, where P is a positive integer. At present P is limited to being not greater than 8 for demonstration purpose. It can be changed to any number as long as there is sufficient memory storage. After entering the value of P in the **HARTLEY** program, P2 is called to calculate the value of N.

Program **FHT** is the heart of the system. It computes the inverse DHT which is the same as the direct DHT without multiplying by a factor of 1/N. For ease of comprehension of the program, it is translated into a BASIC program as shown in figure 6.3. One point to note is the calculation of the trigonometric functions. The partial tangent instruction (FPTAN) is the only trigonometric function available in the Intel 8087. It computes tanA, where A, in radians, is the top stack element and is between 0 and $\pi/4$. The result is a ratio Y/X, with Y replacing A and X being pushed onto the stack. When $0< A < \pi/4$, with the aid of this function, sinA and cosA can be calculated by $Y/SQRT(X^2 + Y^2)$ and $X/SQRT(X^2 + Y^2)$ respectively [25]. When A is outside its acceptable range, some procedures are used to obtain a suitable range that is inside the 0 to $\pi/4$ range. The program first computes all the angles required for the FHT and reduces them to the range between 0 and $\pi/4$, i.e. MOD($\pi/4$). It also keeps

```
10 GOSUB 60    'bit-reversal rtn
20 GOSUB 220    'cal. angles required
30 GOSUB 350   'cal. FHT
40 END
50 'BIT-REVERSAL RTN
60 FOR I=0 TO L-1
70 K=0
80 J=I
90 FOR Q=1 TO P
100 H=J
110 K=K*2+(J-2*H)
120 J=H
130 NEXT Q
140 IND(I)=K
150 NEXT I
160 FOR I=0 TO L-1
170 Y(I)=X(IND(I))
180 NEXT I
190 FOR I=0 TO L-1 : X(I)=Y(I) : NEXT I
200 RETURN
210 ' CAL. ANGLES REQUIRED
220 K=1 : BX=0 : P2=6.283185 : J=P
230 N2=K : K=K+K : FK=P2/K
240 R=N2 : I=0
250 F=FK*I
260 ANG(BX)=F
270 BX=BX+1
280 I=I+1 : R=R-1 : IF R<>0 THEN 250
290 J=J-1 : IF J<>0 THEN 230
300 Q=BX : BX=0
310 C(BX)=COS(ANG(BX)) : S(BX)=SIN(ANG(BX))
320 BX=BX+1 : Q=Q-1 : IF Q<>0 THEN 310
330 RETURN
340 REM  FHT TRANSFORM SECTION
350 K=1 : BX=0 : J=P
360 GOSUB 500
370 N2=K : K=K+K : NM = L-K : Q=NM\K+1
380 R=N2 : I=0
390 T=Q : BP=I : SI=I+N2 : DI=K -I
400 U=X(SI)*C(BX) + X(DI)*S(BX)
410 W=X(BP)
420 Y(BP)=W+U
430 Y(SI)=W-U
440 BP=BP+K : T=T-1 : SI=SI+K : DI=DI+K
450 IF T<>0 THEN 400
460 BX=BX+1 : I=I+1 : R=R-1 : IF R<>0 THEN 390
470 J=J-1 : IF J<>0 THEN 360
480 FOR M=0 TO L-1 : X(M)=Y(M)/L : NEXT M
490 RETURN
500 FOR M=0 TO L-1 : X(M) = Y(M) : NEXT M
510 RETURN
```

Figure 6.3 Program FHT in BASIC

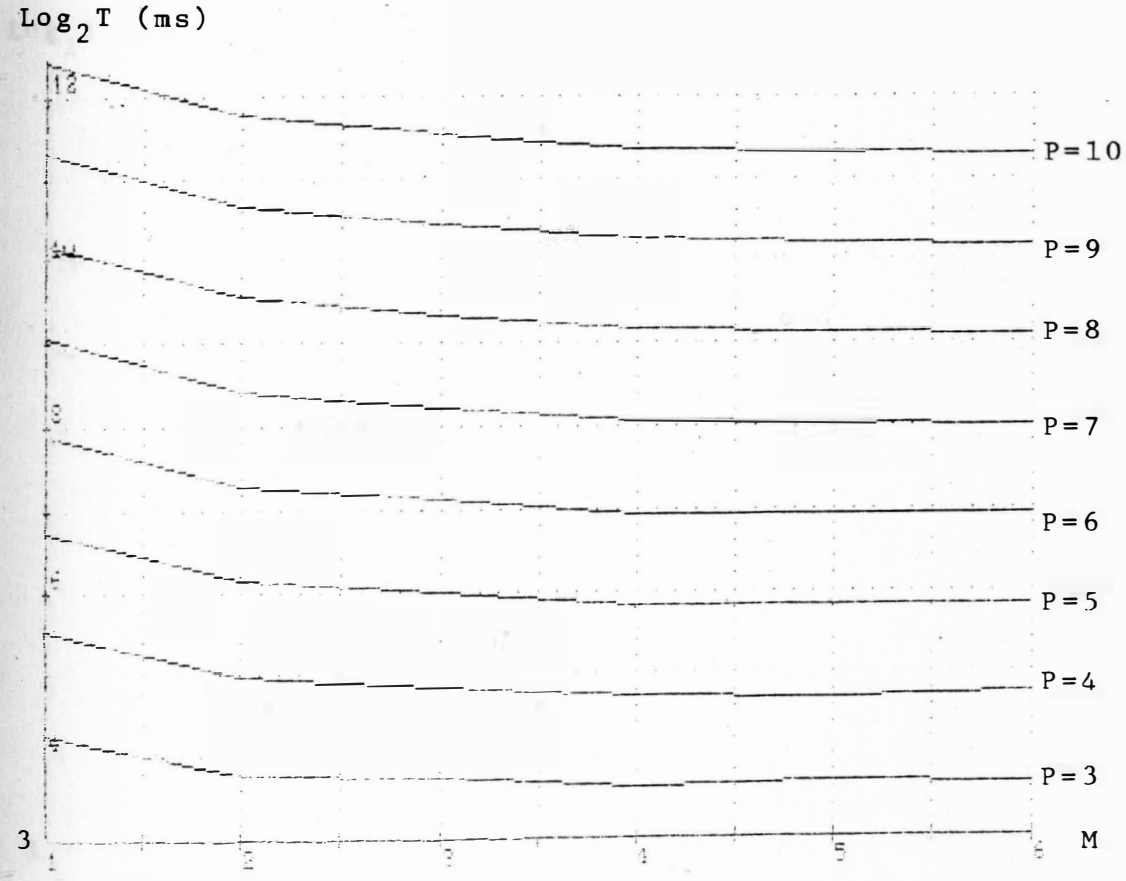


Figure 6.5 Run times for the inverse DHT

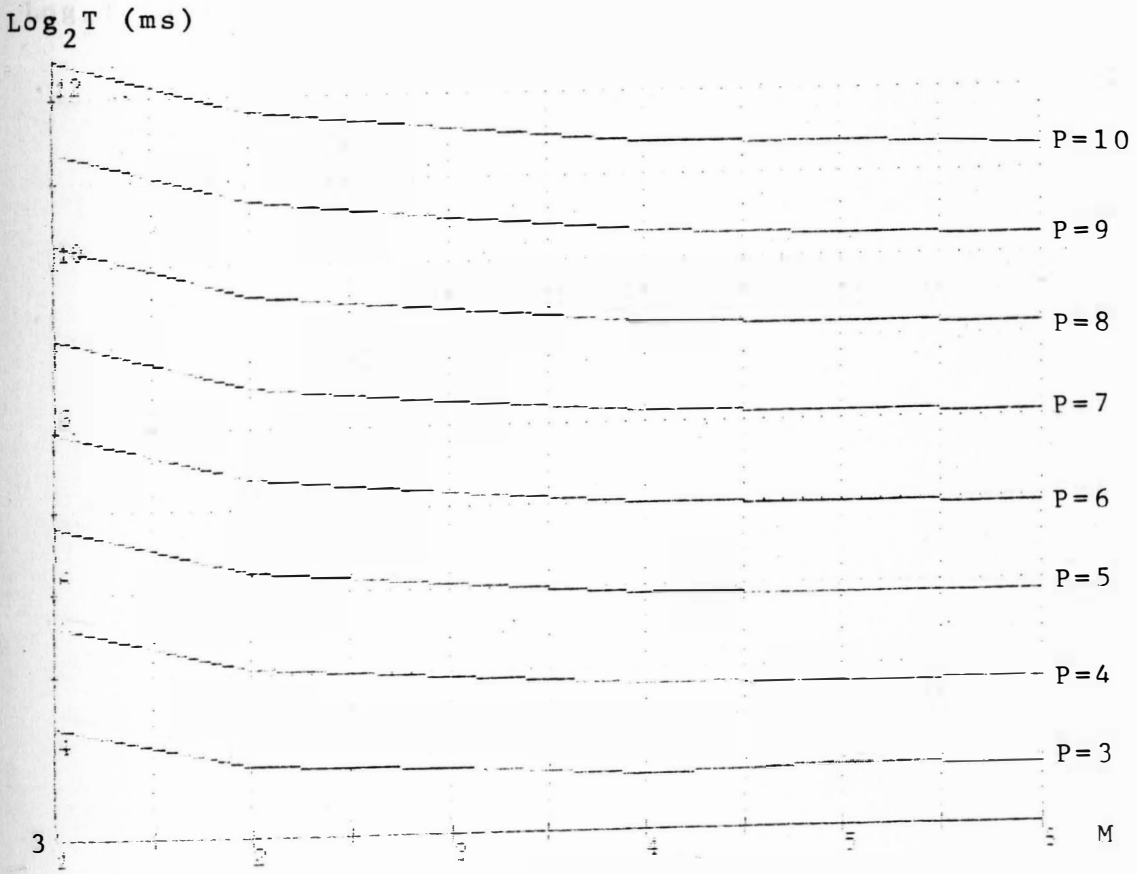


Figure 6.6 Run times for the DHT

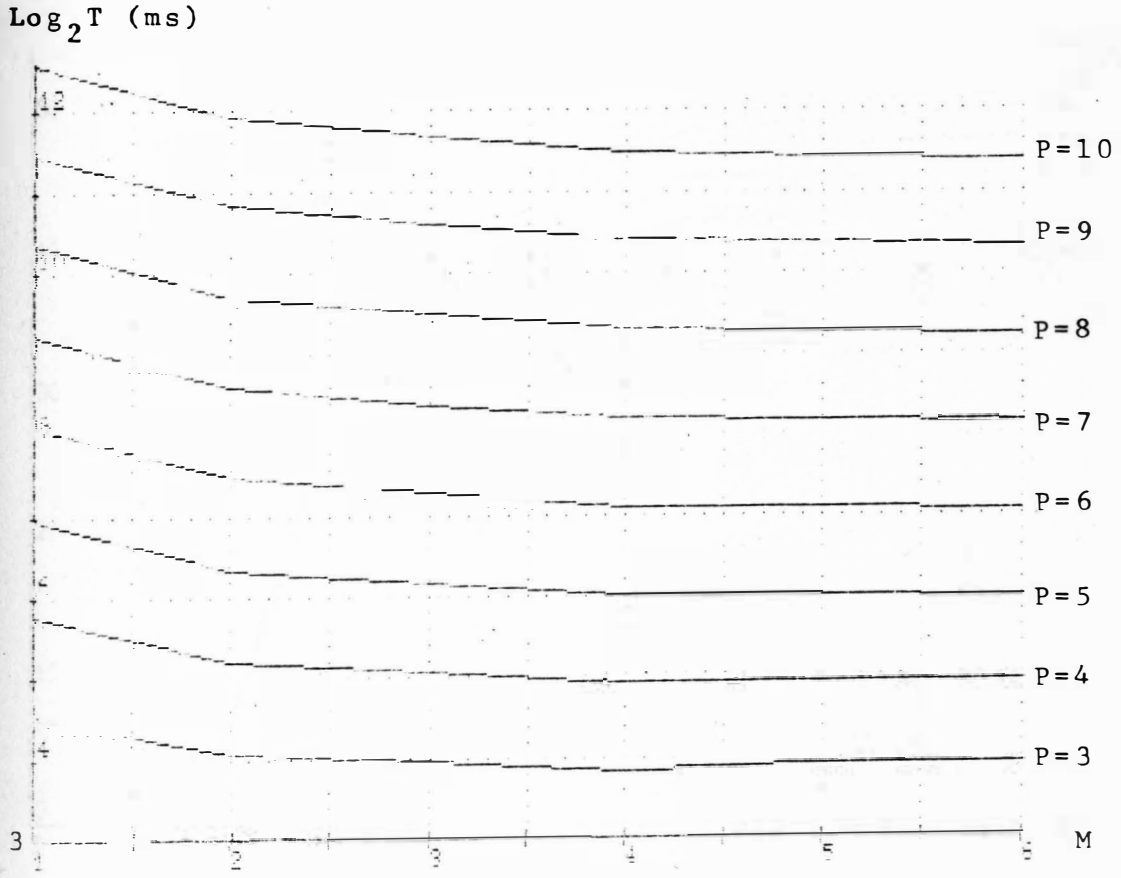


Figure 6.7 Run times for the DFT

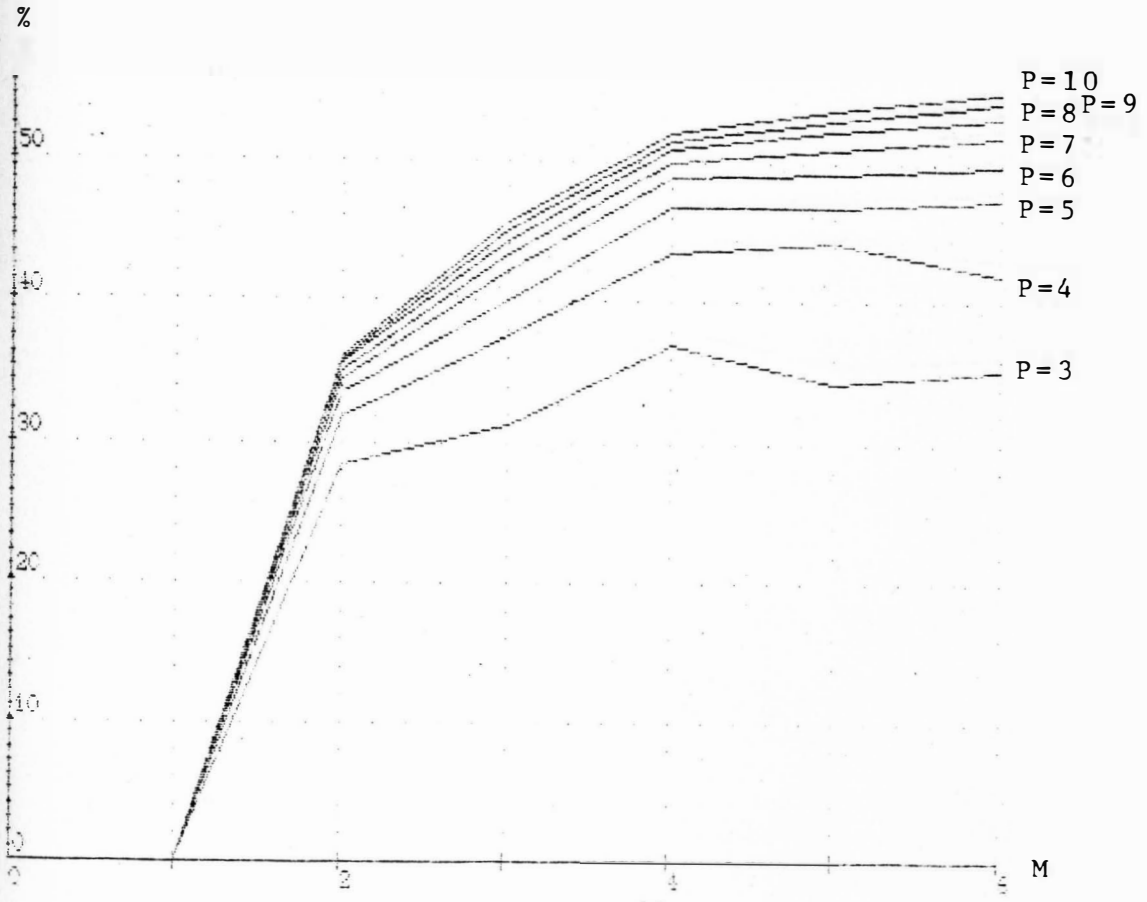


Figure 6.8 Improvement factor for the inverse DHT

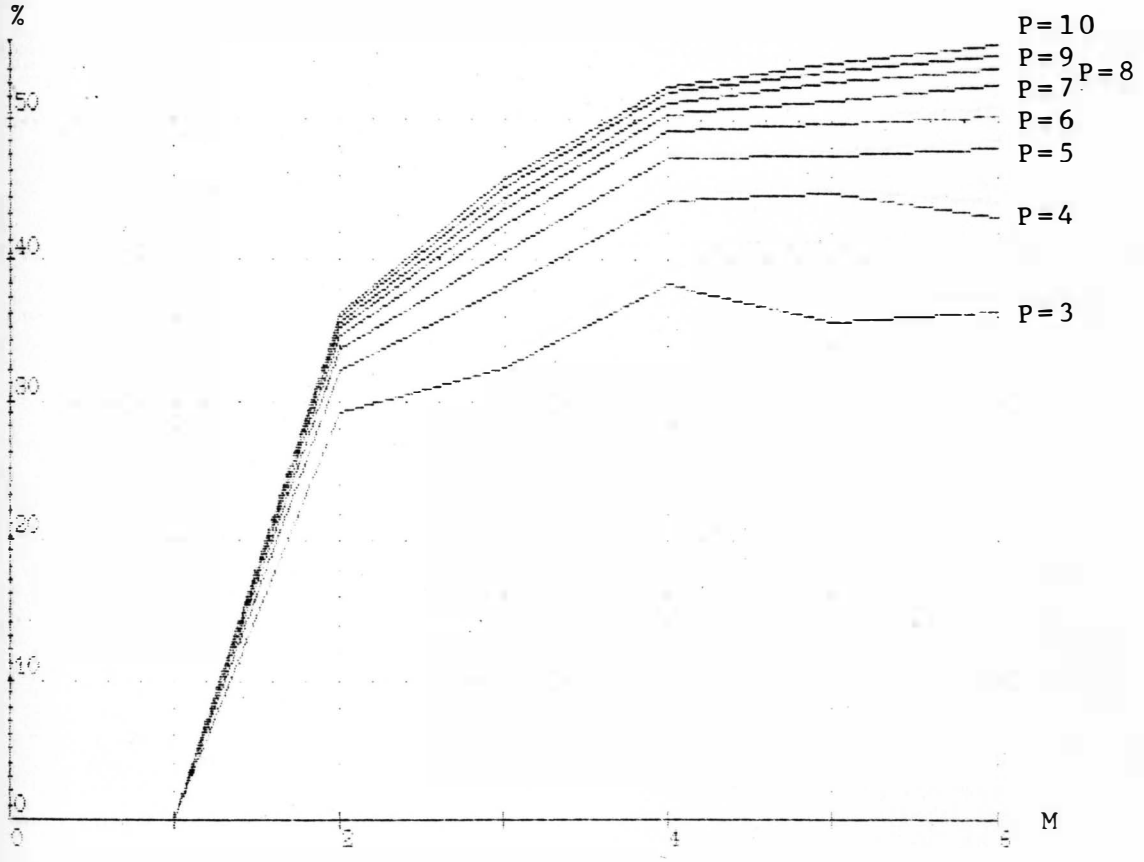


Figure 6.9 Improvement factor for the DHT

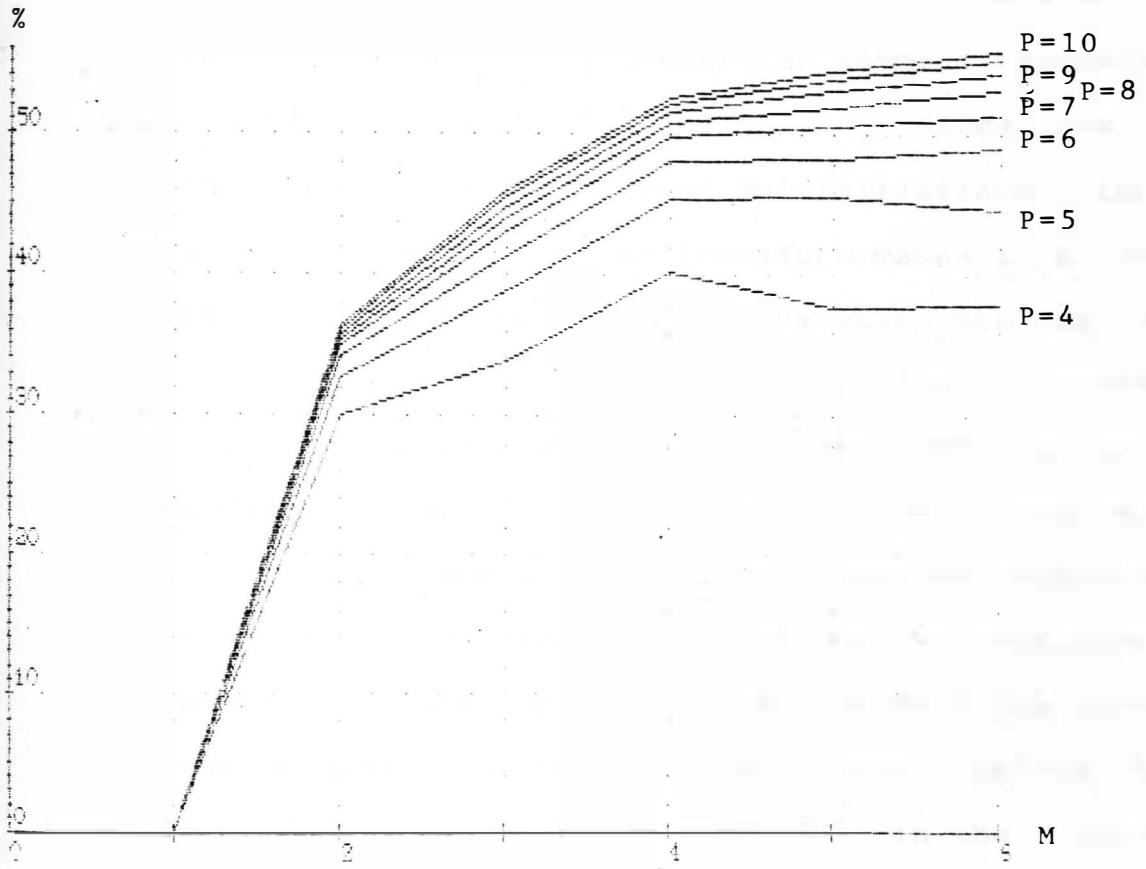


Figure 6.10 Improvement factor for the DFT

CHAPTER SEVEN

CONCLUSION

The Hartley transform is closely related to the Fourier transform. Its properties commend themselves for application to numerical analysis. Since one complex multiplication equals four real multiplications, the real-valued kernel of the Hartley transform makes it a potential replacement to the Fourier transform which has the complex-valued kernel. Most of the existing fast algorithms for computing the discrete Fourier transform (DFT) can be applied to the discrete Hartley transform (DHT) with some modifications. However, the speed of a fast Hartley transform (FHT) should be about twice that of a fast Fourier transform (FFT). In addition, one program is required for both the forward FHT and the inverse. In the FFT, additional control must be implemented for reversing the sign of i in the exponential $\exp(-i2\pi kn/N)$ during the operation of changing from a forward transformation to an inverse transformation or vice-verse. The FHT can also serve as the intermediate stage for the computation of other transforms such as the DFT and the discrete cosine transform (DCT) [13] so that time reduction can be achieved. Although the Hartley transform is a new term in signal processing, many researches on this subject have been carried out [16]-[20]. The growing interest will generate more researches and publications in the near future.

The vector processor (VP) discussed here is a single-instruction, multiple-data stream computer system. The purpose of this device is to speed up the processing of a group of scientific calculations. The greatest improvement can be obtained by minimizing the total data transfer time for each result element. Data independence, avoiding branching on comparison, and reducing data transfer time are the key considerations for programming the parallel processing routine on a VP. Sometimes this is done at the expense of the memory storage. However, the memory storage is relatively less expensive nowadays. A program running on a VP usually consists of scalar mode and parallel mode operations. It should be designed in such a way that scalar mode procedure is at minimum level. If scalar mode procedures dominate the whole program, overall performance of the system will be less significantly improved although parallel mode procedures are at the desired level.

A software system has been designed for computing the DHT, the inverse DHT, and the DFT on a VP. In this particular system, the improvement factor increases with both the number of the data points and the number of the mathematical coprocessors.

Finally, loading a dummy one to the unused MCs is an unproductive process in the VP. Hence it should be eliminated if the improvement of the VP is considered in the future.

REFERENCES

[1] R. V. Hartley, "A more symmetrical Fourier analysis applied to transmission problems," Proc. IRE, vol. 30, pp. 144-150, Mar. 1942.

[2] R. N. Bracewell, "The discrete Hartley transform," J. Opt. Soc. Amer. vol. 73, pp. 1832-1835, Dec. 1983.

[3] ———, "The fast Hartley transform," Proc. IEEE, vol. 72, no. 8, pp. 1010-1018, Aug. 1984.

[4] ———, The Hartley Transforms. New York: Oxford University Press, 1986.

[5] H. J. Meckelburg and D. Lipka, "Fast Hartley transform algorithm," Electron. Lett., vol. 21, pp. 341-343, Apr. 1985.

[6] J. Prado, "Comments on the fast Hartley transform," Proc. IEEE, vol. 73, pp. 1862-1863, Dec. 1985.

[7] G. E. J. Bold, "A comparison of the time involved in computing fast Hartley and fast Fourier transform," Proc. IEEE, vol. 73, pp. 1863-1864.

[8] H. V. Sorensen, D. l. Jones, C. S. Burrus, and M. T. Heidman, "On computing the discrete Hartley transform," IEEE Trans, Acoust., Speech and Signal Processing, vol. ASSP-33, pp. 1231-1238, Oct. 1985.

[9] S. C. Pei and J. L. Wu, "Split-radix fast Hartley transform," Electron. Lett., vol. 22, pp. 26-27, Jan. 1986.

[10] O. Buneman, "Conversion of FFT's to fast Hartley transforms," SIAM J. Sci. Stat. Comput., vol. 7, pp. 624-638, Apr. 1986.

[11] A. Erdèlyi, Tables of Integral Transforms, Vol. 1. New York: McGraw-Hill, 1954.

[12] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput., vol. 19, pp. 297-301, 1965.

[13] H. S. Hou, "The fast Hartley transform algorithm," IEEE Trans. Comput., vol. C-36, no. 2, pp. 147-156, Feb. 1987.

[14] S. Winograd, "On computing the discrete Fourier transform," Math. Comput., vol. 32, no. 141, pp. 175-199, Jan. 1978.

[15] G.D. Berglan and M.T. Dolan, "Subroutine FFA," in Programs for Digital Signal Processing. New York: IEEE PRESS, 1979, pp. 1.2-11 to 1.2-16.

[16] K.H. Tzou and T.R. Hsing, "A study of the discrete Hartley transform for image compression application," Proc. SPIE, vol. 534, Jan. 1985.

[17] A. Zakhor, "Error properties of Hartley transform algorithms," S.M. thesis, Dept. Elec. Eng. Comput. Sci., M.I.T., Cambridge, MA, June 1985.

[18] H. Malvar, "Fast computation of discrete consine transform through fast Hartley transform," Electron. Lett. vol. 22, pp. 352-353, Mar. 1986.

[19] R.N. Bracewell, O. Buneman, and H. Hao, "Fast two-dimensional Hartley transform," Proc. IEEE, vol. 74, no. 9, pp. 1282-1283, Sept. 1986.

[20] A Zakhor and A.V. Oppenheim, "quantization Errors in the Computation of the discrete Hartley transform," IEEE Trans. Acoust., Speech and Signal Processing, vol. ASSP-35, no. 11, pp. 1592-1602, Nov. 1987.

[21] D.B. Miron, "The vector processor; low-cost parallel computing," Proc. NDAS, vol. 41, pp. 84, Apr. 1987.

[22] R.K. Manja, "The vector processor," M.S. thesis, Dept. Elec. Eng., S.D.S.U., Brookings, SD, Dec 1986.

[23] G.J Lipovski and M. Malek, Parallel Computing - Theory and Comparisons. New York: John Wiley & sons, 1987.

[24] Intel Corp., Intel iAPx 86,88,186 and 188 User's Manual Programmer's Reference. California: Intel Corp., 1985.

[25] Y.C. Liu and G.A. Gibson, Microcomputer System: The 8086/8088 Family Architecture, Programming, and Design (2nd Ed.). NJ: Prentice-Hall, 1986.

[26] Brigham, The Fast Fourier Transform. NJ: Prentice-Hall, 1974.

[27] Microsoft Corp., MS Macro Assembler 5.0 - Mixed Language Programming Guide. WA: Microsoft Corp., 1987.

[28] Microsoft Corp., MS QuickBASIC Compiler for IBM PCs and Compatibles. WA: Microsoft Corp., 1985.

LIST OF SYMBOLS

DFT	discrete Fourier transform
DHT	discrete Hartley transform
DIF	decimation-in-frequency
DIT	decimation-in-time
FFT	fast Fourier transform
FHT	fast Hartley transform
Hz	Hertz
GPP	general purpose processor
MC	mathematical coprocessor
SR	split-radix
VCU	vector control unit
VP	vector processor

APPENDIX

COMPUTER PROGRAMS

A software system which consists of one compiled BASIC program and five assembly language programs is designed for computing the fast Hartley transform (FHT) on a vector processor (VP). The block diagram of the system is shown in figure 6.2. Their source programs are listed in this appendix and can be found as follows:

HARTLEY - Figure A-1 ,

FHT - Figure A-2 ,

P2 - Figure A-3 ,

SCALE - Figure A-4 ,

DFT - Figure A-5 ,

SEQ - Figure A-6 .

At present the number of the data points N is limited to 256 for demonstration purpose. However, if the memory storage is sufficient, it can be expanded to any number by modifying the arrays' size in line 250 of **HARTLEY** and the sizes of Y, SIN, COS, ANGLE, and LINDEX as defined in the data segment of the program **FHT**. Note that if the VP is available, NUM_PROC which is defined in **FHT** should be changed to the actual number of the mathematical coprocessors (MCs) and the semi-colon preceded the VP's three instructions should be removed.

```
100 '************************************************'
110 ' This program is to calculate direct and inverse '
120 ' Hartley transform by calling other assembly     '
125 ' language subroutines.                          '
130 ' Assembly language subroutines required:        '
140 '                          P2                    '
150 '                          FHT                   '
160 '                          SEQ                   '
170 '                          SCALE                 '
180 '                          DFT                   '
190 ' Author : Boon Pock Lim                         '
200 ' Program Name : HARTLEY                         '
210 ' Date : March 14, 1988                          '
220 '************************************************'
230 DEFINT N,P
240 DEFSNG I,R,X,Y
250 DIM X(256), Y(256), RE(256), IM(256)
255 '
260 ' MAIN RTN
265 '
280 GOSUB 2100                          'DISPLAY HEADING
290 INPUT "N = 2^P ,    Input P = ",P
300 IF P < 1 OR P > 8 THEN 280
310 CALL P2(P,N)                        'CAL. N
315 LOCATE 5,1 : PRINT SPACE$(10);"P =";P,"N =";N
320 GOSUB 2200                          'INPUT DATA
330 GOSUB 2500                          'SAVE INPUT DATA
340 GOSUB 2600                          'TRANSFORMATION
350 INPUT "New data set ? (Y/N) ",A$
360 IF A$ = "Y" OR A$ = "y" THEN 280
370 END
2095 '
2100 'DISPLAY HEADING
2105 '
2110 CLS
2120 LOCATE 1,1 : PRINT "Date : ";DATE$
2130 LOCATE 1,35 : PRINT "S  D  S  U"
2140 LOCATE 2,26 : PRINT "Electrical Engineering Dept."
2150 LOCATE 4,30 : PRINT "The Hartley Transform"
2160 RETURN
2199 '
2200 ' INPUT DATA
2205 '
2210 LOCATE 7,20 : PRINT "Input data set"
2220 LOCATE 9,20 : PRINT "(1) Auto"
2230 LOCATE 11,20 : PRINT "(2) Manual"
2240 LOCATE 13,20 : INPUT "Option : ",OPT
2250 IF OPT = 1 OR OPT = 2 THEN 2270
```

Figure A-1 HARTLEY program listing

```
2260 LOCATE 13,1 : PRINT SPACE$(80) : GOTO 2240
2270 ON OPT GOSUB 2300, 2400
2280 RETURN
2295 '
2300 ' AUTO INPUT
2305 '
2310 FOR J = 0 TO N-1
2320 X(J) = J+1
2330 NEXT J
2340 FOR J = 7 TO 13 STEP 2
2350 LOCATE J,1 : PRINT SPACE$(80)
2360 NEXT J
2370 RETURN
2395 '
2400 ' MANUAL INPUT
2405 '
2410 FOR J = 9 TO 13 STEP 2
2420 LOCATE J,1 : PRINT SPACE$(80)
2430 NEXT J
2440 LN = 9
2445 FOR J = 0 TO N-1
2447 IF LN <> 23 THEN 2460
2450 FOR K = 9 TO 23
2452 LOCATE K,1 : PRINT SPACE$(80)
2454 NEXT K
2456 LN = 9
2460 LOCATE LN,1 : PRINT "X (";J;") = "; : INPUT "",X(J)
2470 LN = LN + 1
2480 NEXT J
2482 FOR J = 7 TO LN
2484 LOCATE J,1 : PRINT SPACE$(80)
2486 NEXT J
2490 RETURN
2495 '
2500 ' SAVE INPUT DATA
2505 '
2510 FOR J = 0 TO N-1
2520 Y(J) = X(J)
2530 NEXT J
2540 RETURN
2595 '
2600 ' TRANSFORMATION
2605 '
2610 LOCATE 7,20 : PRINT "(1) Direct FHT"
2620 LOCATE 9,20 : PRINT "(2) Inverse FHT"
2630 LOCATE 11,20 : INPUT "Option : ",OPT
2640 IF OPT = 1 OR OPT = 2 THEN 2660
2650 LOCATE 11,1 : PRINT SPACE$(80) : GOTO 2630
```

Figure A-1 (continue)

```
2660 ON OPT GOSUB 2700, 2800
2665 GOSUB 2900                              'OUTPUT
2670 RETURN
2695 '
2700 ' DIRECT FHT
2705 '
2710 CALL FHT(X(0),P)
2720 CALL SCALE(X(0),N)
2730 CALL DFT(X(0),N,RE(0),IM(0))
2740 RETURN
2795 '
2800 ' INVERSE FHT
2805 '
2810 CALL FHT(X(0),P)
2820 RETURN
2895 '
2900 ' OUTPUT
2905 '
2910 LOCATE 7,20 : PRINT "Output          "
2920 LOCATE 9,20 : PRINT "(1) Display      "
2930 LOCATE 11,20 : PRINT "(2) Print          "
2940 LOCATE 13,20 : INPUT "Option : ",T
2950 IF T = 1 OR T = 2 THEN 2970
2960 LOCATE 13,1 : PRINT SPACE$(80) : GOTO 2940
2970 FOR J = 9 TO 13 STEP 2
2972 LOCATE J,1 : PRINT SPACE$(80)
2974 NEXT J
2980 ON T GOSUB 3000, 3300
2990 RETURN
2995 '
3000 ' DISPLAY OUTPUT
3005 '
3010 LOCATE 9,1
3020 ON OPT GOSUB 3200,3250                  'SUB HEADING
3040 LN = 9
3050 FOR J = 0 TO N-1
3060 IF LN <> 22 THEN 3110
3070 LOCATE 23,1 : PRINT "PRESS ANY KEY TO CONTINUE"
3080 K$ = INKEY$ : IF K$ = "" THEN 3080
3090 FOR K = 10 TO 23
3092 LOCATE K,1 : PRINT SPACE$(80)
3094 NEXT K
3100 LN = 9
3105 LOCATE 10,1
3110 LN = LN + 1
3120 PRINT USING "###";J;TAB(10);
3130 ON OPT GOSUB 3220,3260            'DISPLAY OUTPUT
3150 NEXT J
```

Figure A-1 (continue)

```
3160 RETURN
3195 '
3200 ' DISPLAY DIRECT FHT SUB HEADING
3205 '
3207 PRINT " n,k";TAB(18);"f(n)";TAB(30);"H(k)";
3210 PRINT TAB(41);"RE[F(k)]";TAB(54);"IM[F(k)]"
3212 RETURN
3215 '
3220 ' DISPLAY DIRECT FHT
3222 '
3230 PRINT USING " ####.#######";Y(J),X(J),RE(J),IM(J)
3232 RETURN
3245 '
3250 ' DISPLAY INVERSE FHT SUB HEADING
3252 '
3254 PRINT " n,k";TAB(18);"H(k)";TAB(30);"f(n)"
3256 RETURN
3258 '
3260 ' DISPLAY INVERSE FHT
3262 '
3264 PRINT USING " ####.#######";Y(J),X(J)
3266 RETURN
3295'
3300 ' PRINT OUTPUT
3305 '
3310 PG = 0 : GOSUB 4000                    'PRINT HEADING
3320 FOR J = 0 TO N-1
3330 IF LN > 56 THEN GOSUB 4000
3340 LN = LN + 1
3350 LPRINT USING "###";J;TAB(10);
3360 ON OPT GOSUB 4150,4250                 'PRINT OUTPUT
3380 NEXT J
3390 RETURN
3995 '
4000 ' PRINT HEADING
4005 '
4010 LPRINT CHR$(12) : PG = PG + 1 : LPRINT
4020 LPRINT "Date : ";DATE$;TAB(35);"S  D  S  U";TAB(71);"Page";PG
4030 LPRINT TAB(26);"Electrical Engineering Dept."
4040 LPRINT
4045 LPRINT "The Hartley Transform :    P =";P;" ,  N = 2^P =";N
4047 LPRINT
4050 ON OPT GOSUB 4100,4200                 'SUB HEADING
4070 LN = 7
4080 RETURN
4095 '
4100 ' PRINT DIRECT FHT SUB HEADING
4105 '
```

Figure A-1 (continue)

```
4110 LPRINT " n,k";TAB(18);"f(n)";TAB(30);"H(k)";
4115 LPRINT TAB(41);"RE[F(k)]";TAB(54);"IM[F(k)]"
4120 RETURN
4145 '
4150 ' PRINT DIRECT FHT
4155 '
4160 LPRINT USING " ####.#######";Y(J),X(J),RE(J),IM(J)
4170 RETURN
4195 '
4200 ' PRINT INVERSE FHT SUB HEADING
4205 '
4210 LPRINT " n,k";TAB(18);"H(k)";TAB(30);"f(n)"
4220 RETURN
4245 '
4250 ' PRINT INVERSE FHT
4255 '
4260 LPRINT USING " ####.#######";Y(J),X(J)
4270 RETURN
```

Figure A-1 (continue)

```
.8087
.MODEL  MEDIUM
.CODE
;-------------------------------------------------------;
; Program Name : FHT                                    ;
; This subroutine(Procedure) calculates the Fast Hartley ;
; Transform by using radix-2 method (decimation in time). ;
; From compiled BASIC program : CALL FHT(X(0),P)        ;
; Subroutines(Procedures) required : BIT_REV            ;
;                                    REORDER_Y          ;
;                                    Y_TO_X             ;
;                                    ANGLE_TB           ;
;                                    H_TRAN             ;
; Author : Boon Pock Lim                                ;
; Date   : Feb 22, 1988                                 ;
;-------------------------------------------------------;
                EXTRN   SEQ_LD_2:FAR
                EXTRN   SEQ_ST_2:FAR
                EXTRN   SEQ_LD_4:FAR
                EXTRN   SEQ_ST_4:FAR

                PUBLIC  FHT
FHT             PROC    FAR
                PUSH    BP
                MOV     BP,SP
                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX
                PUSH    DX              ;------------------
                MOV     BX,[BP]+8       ;1st Arg - Store the
                MOV     X,BX            ;address in X
                MOV     BX,[BP]+6       ;2nd Arg
                MOV     CX,[BX]
                MOV     POWER,CX
                MOV     AX,1            ;Calculate # of
                SHL     AX,CL           ;elements N
                MOV     N,AX            ;N = 2^POWER
                MOV     CX,AX           ;Set CX = N
                XOR     BX,BX           ;Set BX = 0
                XOR     DX,DX           ;Set DX = 0
MAIN_1:
                MOV     LINDEX[BX],DX   ;Set up index table
                ADD     BX,2            ;0,1,2,.....,N-1
                INC     DX
                LOOP    MAIN_1          ;------------------
;
;               JMP     MAIN_D_1        ;Dummy jump
;MAIN_D_1:
```

Figure A-2 FHT program listing

```
;                DB      0DFh,0FDh       ;Parallel mode
;
                 MOV     DX,POWER        ;Set DX = POWER
                 CALL    BIT_REV         ;Bit reversing
                 CALL    REORDER_Y       ;Y = new order of X
                 CALL    Y_TO_X          ;Set X = Y
                 CALL    ANGLE_TB        ;set up angle table
                 CALL    H_TRAN          ;FHT
;
;                JMP     MAIN_D_2        ;Dummy jump
;MAIN_D_2:
;                DB      0DFh,0FEh       ;Scalar mode
;
                 POP     DX              ;Restore the
                 POP     CX              ;registers
                 POP     BX
                 POP     AX
                 POP     BP              ;-----------------
                 RET     4
FHT              ENDP


                 PUBLIC  BIT_REV
BIT_REV          PROC    NEAR
;-------------------------------------------------;
; This procedure performs bit reversing operation ;
; Input parameters : AX = # of elements           ;
;                    DX = # of bits               ;
; Reads : LINDEX, BASE, TEMP                      ;
; Writes : LINDEX, TEMP                           ;
; Algorithm :   Let J be the original index       ;
;               K = 0                             ;
;               FOR Q = 1 TO (# of bits)          ;
;                       L = J MOD 2               ;
;                       J = INT(J/2)              ;
;                       K = K * 2 + L             ;
;               NEXT Q                            ;
;               new index = K                     ;
; Contents of the 8087's registers :              ;
;               ST(7) = -1                        ;
;               ST(6) = J = original index        ;
;               ST(5) = 2 = BASE                  ;
;               ST(4) = L = J MOD 2               ;
;               ST(3) = 1                         ;
;               ST(2) = K                         ;
;-------------------------------------------------;
                 PUSH    AX              ;Save the registers
                 PUSH    BX
```

Figure A-2 (continue)

```
                PUSH    CX
                PUSH    DX              ;------------------
                DEC     AX              ;AX = N - 2 , 1st
                DEC     AX              ;and last elements
                OR      AX,AX           ;are the same.
                JNZ     BIT             ;JZ
                JMP     BIT_4
BIT:
                MOV     BX,2            ;Begin with 2nd
BIT_1:                                  ;element
                POP     DX              ;Get # of bits
                PUSH    DX              ;Store
                FINIT                   ;Initialize 8087
                FSTCW   TEMP            ;Set 8087 rounding
                FWAIT                   ;control -
                OR      TEMP,0C00h      ;round down
                FLDCW   TEMP
                FLD1                    ;ST(7)=-1 , Scale
                FCHS                    ;factor
                PUSH    BX              ;Load index
                LEA     BX,LINDEX[BX]   ;sequentially
                CALL    SEQ_LD_2
                POP     BX              ;-----------------
                FILD    BASE            ;ST(5)=2 , Base
                FDECSTP                 ;SET ST=4
                FLD1                    ;ST(3)=1
                FLDZ                    ;ST(2)=K ,
BIT_2:                                  ;init. value = 0
                FINCSTP                 ;SET ST=4
                FINCSTP
                FSTP    ST(0)           ;-----------------
                FLD     ST(1)           ;ST(4)=L=J
                FPREM                   ;L=J MOD 2
                FINCSTP                 ;SET ST=6
                FINCSTP                 ;-----------------
                FSCALE                  ;J=INT(J/2)
                FRNDINT                 ;-----------------
                FDECSTP                 ;SET ST=2
                FDECSTP
                FDECSTP
                FDECSTP                 ;-----------------
                FSCALE                  ;K=K*2
                FADD    ST,ST(2)        ;K=K*2+L
                FWAIT                   ;remainder
                DEC     DX
                OR      DX,DX           ;Do POWER times
                JNZ     BIT_2
                PUSH    BX              ;Store new index
```

Figure A-2 (continue)

```
                LEA     BX,LINDEX[BX]   ;sequentially
                CALL    SEQ_ST_2
                POP     BX              ;-----------------
                MOV     CX,NUM_PROC     ;Set next address
                CMP     CX,AX
                JBE     BIT_3
                MOV     CX,AX
BIT_3:
                ADD     BX,2
                LOOP    BIT_3           ;-----------------
                SUB     AX,NUM_PROC     ;Do (N-2) times
                CMP     AX,0
                JLE     BIT_4           ;JG will cause out
                JMP     BIT_1           ;of range error
BIT_4:
                POP     DX              ;Restore the
                POP     CX              ;registers
                POP     BX
                POP     AX              ;-----------------
                RET

BIT_REV         ENDP


                PUBLIC  REORDER_Y
REORDER_Y       PROC    NEAR
;-------------------------------------------------------------;
; This procedure makes Y equal to bit-reversal order of X.    ;
; Input parameter : AX = N                                    ;
; Reads : LINDEX, X                                           ;
; Writes : Y                                                  ;
;-------------------------------------------------------------;

                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX
                PUSH    SI
                PUSH    DI              ;-------------------
                MOV     CX,AX           ;Set CX = N
                XOR     BX,BX           ;Set BX = 0
                XOR     DI,DI           ;Set DI = 0
;
;               JMP     REORDER_D_1     ;Dummy jump
;REORDER_D_1:
;               DB      0DFh,0FEh       ;Scalar mode
;
                FINIT                   ;Initialize 8087
```

Figure A-2 (continue)

```
REORDER:
                MOV     SI,LINDEX[BX]   ;Find the actual
                ADD     SI,SI           ;address of X
                ADD     SI,SI           ;Offset = index * 4
                ADD     SI,X            ;------------------
                FLD     DWORD PTR [SI]  ;Move X to Y
                FSTP    Y[DI]           ;------------------
                ADD     DI,4            ;Next addr of Y
                ADD     BX,2            ;Next addr of index
                LOOP    REORDER         ;Do N times
;
;               JMP     REORDER_D_2     ;Dummy jump
;REORDER_D_2:
;               DB      0DFh,0FDh       ;Parallel mode
;

                POP     DI              ;Restore the
                POP     SI              ;registers
                POP     CX
                POP     BX
                POP     AX              ;--------------------
                RET
REORDER_Y       ENDP

                PUBLIC  Y_TO_X
Y_TO_X          PROC    NEAR
;-----------------------------------------------------;
; This procedure moves Y to X.                        ;
; Input parameter : AX = N                            ;
; Reads : Y                                           ;
; Writes : X                                          ;
;-----------------------------------------------------;
                PUSH    AX              ;Save the registers
                PUSH    CX
                PUSH    SI
                PUSH    DI              ;------------------
                MOV     CX,AX           ;Set CX = N
                XOR     SI,SI           ;Set SI = 0
                MOV     DI,X            ;DI = 1st addr of X
;
;               JMP     Y_X_D_1         ;Dummy jump
;Y_X_D_1:
;               DB      0DFh,0FEh       ;Scalar mode
;
                FINIT                   ;Initialize 8087
Y_X_MOVE:
                FLD     Y[SI]           ;Transfer Y to X
```

Figure A-2 (continue)

```
                FSTP    DWORD PTR [DI]  ;-------------------
                ADD     SI,4            ;Next addr - 2 for
                ADD     DI,4            ;word, 4 for S. real
                LOOP    Y_X_MOVE        ;and 8 for D. real
;
;               JMP     Y_X_D_2         ;Dummy jump
;Y_X_D_2:
;               DB      0DFh,0FDh       ;Parallel mode
;
                POP     DI              ;Restore the
                POP     SI              ;registers
                POP     CX
                POP     AX              ;-------------------
                RET
Y_TO_X          ENDP


                PUBLIC  ANGLE_TB
ANGLE_TB        PROC    NEAR
;-----------------------------------------------------------;
; This procedure calculates sine and cosine of the angles  ;
; required for the FHT.                                     ;
; Reads : POWER, ANGLE, LINDEX, SIN, COS                    ;
; Writes : ANGLE, LINDEX, SIN, COS                          ;
; Intermediate variables : K, N2, I                         ;
; Refer to the BASIC program for more information           ;
;-----------------------------------------------------------;
                PUSH    AX              ;Store the register
                PUSH    BX
                PUSH    CX
                PUSH    DX
                PUSH    SI              ;------------------
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^;
; This routine calculates all the angles required for the   ;
; FHT and reduce them to the range between 0 and PI/4       ;
; i.e. MOD PI/4.  It also calculates the corresponding      ;
; octant (0 - 7) of the angles.                             ;
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^;
                MOV     K,1             ;Set K = 1
                XOR     BX,BX           ;Set BX = 0
                XOR     SI,SI           ;Set SI = 0
                FINIT                   ;Initialize 8087
                FLD1                    ;ST(7)=1
                FLDPI                   ;ST(6)=PI
                FSCALE                  ;ST(6)=2*PI
                FSTP    ST(1)           ;ST(7)=2*PI
                FILD    BASE            ;ST(6)=2
                FCHS                    ;ST(6)=-2
```

Figure A-2 (continue)

```
                FLDPI                   ;ST(5)=PI
                FSCALE                  ;ST(5)=PI/4
                FSTP    ST(1)           ;ST(6)=PI/4
                FWAIT
                XOR     DI,DI           ;Set DI=0
                MOV     CX,POWER        ;1st loop, CX=POWER
ANG_POWER:
                MOV     AX,K            ;N2 = K
                SHL     K,1             ;K = K + K
                FILD    K               ;ST(5)=K
                FDIVR   ST,ST(2)        ;ST(5)=2*PI/K
                FWAIT
                XOR     DX,DX           ;Set DX = 0
ANG_I_LP:
                CALL    ANG_LD_I        ;Load I sequentially
                FMUL    ST,ST(1)        ;ST(4)=2*PI*I/K
                FLD     ST(0)           ;ST(3)=ST(4)
                FINCSTP                 ;Set ST=4
                FXCH    ST(2)           ;ST(4)=PI/4
                FDECSTP                 ;Set ST(3)
                FPREM                   ;ST(3)=ST(3) MOD PI/4
                PUSH    BX              ;Store ANGLE
                                        ;sequentially
                LEA     BX,ANGLE[BX]    ;0 <= ANGLE <= PI/4
                CALL    SEQ_ST_4
                POP     BX              ;-----------------
                FXCH    ST(2)           ;Exchange
                FINCSTP                 ;Set ST=5
                FXCH    ST(2)           ;ST(5)=2*PI
                FDECSTP                 ;Set ST=4
                FPREM                   ;ST(4)=ST(4) MOD 2*PI
                FDIV    ST,ST(2)        ;ST(4)=ST(4)/(PI/4)
                FSTCW   TEMP            ;Set 8087 rounding control
                FWAIT                   ;control
                OR      TEMP,0C00h      ;Round down
                FLDCW   TEMP
                PUSH    BX              ;Store the octant
                LEA     BX,LINDEX[SI]   ;(0-7)
                CALL    SEQ_ST_2
                POP     BX
                FXCH    ST(2)           ;Exchange
                AND     TEMP,0F3FFh     ;Set 8087 rounding
                FLDCW   TEMP            ;control - round to
                FWAIT                   ;nearest or even
                PUSH    CX              ;Set next address
                MOV     CX,NUM_PROC
                CMP     CX,AX
                JBE     ANG_INC_1
```

Figure A-2 (continue)

```
                MOV     CX,AX
ANG_INC_1:
                ADD     BX,4
                ADD     SI,2            ;-----------------
                INC     DI              ;DI=DI+1
                INC     DX              ;DX=DX+1
                LOOP    ANG_INC_1
                POP     CX
                MOV     I,DX            ;I=DX
                SUB     AX,NUM_PROC
                CMP     AX,0
                JLE     ANG_A
                JMP     ANG_I_LP
ANG_A:
                FSTP    ST(0)
                FWAIT
                DEC     CX              ;LOOP will cause out
                OR      CX,CX           ;of range error
                JZ      ANG_NEXT        ;-------------------
                JMP     ANG_POWER       ;1st loop end
ANG_NEXT:
                MOV     DX,DI           ;DX=# of elements in
                                        ;ANGLE
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^;
; This routine changes the angle A in octant 1,3,5,7 to    ;
; (PI/4 - A).                                               ;
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^;
;
;               JMP     ANG_D_1         ;Dummy jump
;ANG_D_1:
;               DB      0DFh,0FEh       ;Scalar mode
;
                FINIT                   ;Reset 8087
                FILD    BASE            ;ST(7)=2
                FCHS                    ;ST(7)=-2
                FLDPI                   ;ST(6)=PI
                FSCALE                  ;ST(6)=PI/4
                FSTP    ST(1)           ;ST(7)=PI/4
                XOR     BX,BX           ;BX=0
                XOR     SI,SI           ;SI=0
                MOV     CX,DX           ;CX=DX
ANG_ADJ:                                ;Loop begin
                MOV     AX,LINDEX[SI]   ;Check the octant
                SHR     AX,1            ;(PI -angle) if it
                JNC     ANG_OUT         ;is odd
                FLD     ANGLE[BX]
                FSUBR   ST(0),ST(1)
                FSTP    ANGLE[BX]       ;-------------------
```

Figure A-2 (continue)

```
                FWAIT
ANG_OUT:
                ADD     BX,4            ;Set next addresses
                ADD     SI,2            ;------------------
                LOOP    ANG_ADJ         ;Loop end
;
;               JMP     ANG_D_2         ;Dummy jump
;ANG_D_2:
;               DB      0DFh,0FDh       ;Parallel mode
;
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^;
; This routine calculates sine and cosine of the angle                    ;
; with a proper sign.                                                     ;
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^;
                XOR     BX,BX           ;BX=0
                XOR     SI,SI           ;SI=0
                MOV     AX,DX           ;AX=DX
ANG_CAL:
                FINIT                   ;Reset 8087
                FLD1                    ;ST(7)=Sign of sine
                PUSH    BX              ;=1
                MOV     BX,SI
                CALL    SIGN_S
                POP     BX
                FLD1                    ;ST(6)=Sign of
                PUSH    BX              ;cosine=1
                MOV     BX,SI
                CALL    SIGN_C
                POP     BX
ANG_CONT:
                PUSH    BX              ;Load ANGLE
                LEA     BX,ANGLE[BX]    ;sequentially
                CALL    SEQ_LD_4
                POP     BX
                FPTAN                   ;ST(5)=Y, ST(4)=X
                FMUL    ST,ST(2)        ;Set sign for cosine
                FXCH    ST(1)           ;ST(5)=X, ST(4)=Y
                FMUL    ST,ST(3)        ;Set sign for sine
                FLD     ST(0)           ;ST(3)=ST(4)=Y
                FMUL    ST,ST(1)        ;ST(3)=Y*Y
                FLD     ST(2)           ;ST(2)=X
                FMUL    ST,ST(3)        ;ST(2)=X*X
                FADD    ST,ST(1)        ;ST(2)=(X*X) + (Y*Y)
                FSQRT                   ;ST(2)=SQRT(ST(2))
                FLD     ST(0)           ;ST(1)=ST(2)
                FDIVR   ST,ST(3)        ;ST(1)=ST(4)/ST(1)
                PUSH    BX              ;=sine
                LEA     BX,SIN[BX]      ;Store SIN
```

Figure A-2 (continue)

```
                CALL    SEQ_ST_4        ;sequentially
                POP     BX              ;----------------
                FDIVR   ST,ST(3)        ;ST(2)=ST(5)/ST(2)
                PUSH    BX              ;=cosine
                LEA     BX,COS[BX]      ;Store COS
                CALL    SEQ_ST_4        ;sequentially
                POP     BX              ;----------------
                PUSH    CX
                MOV     CX,NUM_PROC
                CMP     CX,AX
                JBE     ANG_INC_2
                MOV     CX,AX
ANG_INC_2:
                ADD     BX,4            ;Set next addresses
                ADD     SI,2            ;----------------
                LOOP    ANG_INC_2
                POP     CX
                SUB     AX,NUM_PROC
                CMP     AX,0
                JG      ANG_CAL         ;Loop end
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^;
; This routine swaps the values of sine and cosine for                    ;
; the angle in octant 1,2,5,6.                                            ;
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^;
;
;               JMP     ANG_D_3         ;Dummy jump
;ANG_D_3:
;               DB      0DFh,0FEh       ;Scalar mode
;
                FINIT                   ;Reset 8087
                XOR     BX,BX           ;BX=0
                XOR     SI,SI           ;SI=0
                MOV     CX,DX           ;CX=DX
ANG_SWAP:                               ;Loop begin
                CMP     LINDEX[SI],0    ;Swap sin and cos
                JE      ANG_DONE        ;if octant is
                CMP     LINDEX[SI],3    ;1,2,5,6
                JE      ANG_DONE
                CMP     LINDEX[SI],4
                JE      ANG_DONE
                CMP     LINDEX[SI],7
                JE      ANG_DONE
                FLD     SIN[BX]
                FLD     COS[BX]
                FSTP    SIN[BX]
                FSTP    COS[BX]         ;----------------
ANG_DONE:
                ADD     BX,4            ;Set next addresses
```

Figure A-2 (continue)

```
                ADD     SI,2            ;------------------
                LOOP    ANG_SWAP        ;Loop end
;
;               JMP     ANG_D_4         ;Dummy jump
;ANG_D_4:
;               DB      0DFh,0FDh       ;Parallel mode
;
                POP     SI              ;Restore the
                POP     DX              ;registers
                POP     CX
                POP     BX
                POP     AX              ;------------------
                RET
ANGLE_TB        ENDP

ANG_LD_I        PROC    NEAR
;-------------------------------------------------;
; This procedure loads integer I sequentially.    ;
; Input parameters : AX = # of data to be processed ;
;                    DX = 1st # of I              ;
; Reads : NUM_PROC                                ;
;-------------------------------------------------;
                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX
                PUSH    DX
                PUSH    SI              ;------------------
;
;               JMP     ANG_LD_D1       ;Dummy jump
;ANG_LD_D1:
;               DB      0DFh,0FFh       ;Sequential mode
;
                MOV     BX,DX           ;BX=DX
                XOR     DX,DX           ;If AX < # of
                MOV     CX,NUM_PROC     ;processors then
                MOV     SI,CX           ;CX=AX else
                CMP     CX,AX           ;CX=# or processors
                JBE     ANG_LD_1        ;DX : Flag
                MOV     CX,AX
                MOV     DX,1
ANG_LD_1:
                MOV     I,BX            ;Set I
                FILD    I               ;Load I
                INC     BX              ;BX=BX+1
                LOOP    ANG_LD_1
                OR      DX,DX
                JZ      ANG_LD_3
                MOV     CX,SI
```

Figure A-2 (continue)

```
                SUB     CX,AX
ANG_LD_2:
                FLD1                    ;Load dummy 1 to
                LOOP    ANG_LD_2        ;other processors.
ANG_LD_3:
;
;               JMP     ANG_LD_D2       ;Dummy jump
;ANG_LD_D2:
;               DB      0DFh,0FEh       ;Scalar mode
;               JMP     ANG_LD_D3       ;Dummy jump
;ANG_LD_D3:
;               DB      0DFh,0FDh       ;Parallel mode
;
                POP     SI              ;Restore the
                POP     DX              ;registers
                POP     CX
                POP     BX
                POP     AX              ;---------------
                RET
ANG_LD_I        ENDP


                PUBLIC  SIGN_S
SIGN_S          PROC    NEAR
;-------------------------------------------------------;
; This procedure checks the octant [LINDEX] sequentially ;
; and changes the sign of ST(7) (sine) if required.      ;
; Input parameters : AX = # of data to be processed      ;
;                    BX = index of LINDEX                ;
; Reads : NUM_PROC, LINDEX                               ;
; Writes : ST(7)                                         ;
;-------------------------------------------------------;
                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX              ;-----------------
;
;               JMP     SIGN_S_D_1      ;Dummy jump
;SIGN_S_D_1:
;               DB      0DFh,0FFh       ;Sequential mode
;
                MOV     CX,NUM_PROC     ;Set the count
                CMP     CX,AX
                JBE     SIGN_S_1
                MOV     CX,AX
SIGN_S_1:
                CMP     LINDEX[BX],2    ;If octant = 0,1,3,6
                JB      SIGN_S_2        ;then sine = +ve
                CMP     LINDEX[BX],3    ;else sine = -ve
```

Figure A-2 (continue)

```
                JE      SIGN_S_2
                CMP     LINDEX[BX],6
                JE      SIGN_S_2
                FCHS                        ;-------------------
SIGN_S_2:
                ADD     BX,2                ;Set next address
                LOOP    SIGN_S_1            ;-------------------
;
;               JMP     SIGN_S_D_2          ;Dummy jump
;SIGN_S_D_2:
;               DB      0DFh,0FEh           ;Scalar mode
;               JMP     SIGN_S_D_3          ;Dummy jump
;SIGN_S_D_3:
;               DB      0DFh,0FDh           ;Parallel mode
;
                POP     CX                  ;Restore the
                POP     BX                  ;registers
                POP     AX
                RET
SIGN_S          ENDP


                PUBLIC  SIGN_C
SIGN_C          PROC    NEAR
;-------------------------------------------------------;
; This procedure checks the octant [LINDEX] sequentially ;
; and changes the sign of ST(6) (cosine) if required.    ;
; Input parameters : AX = # of data to be processed      ;
;                    BX = index of LINDEX                ;
; Reads : NUM_PROC, LINDEX                               ;
; Writes : ST(6)                                         ;
;-------------------------------------------------------;
                PUSH    AX                  ;Save the registers
                PUSH    BX
                PUSH    CX                  ;-----------------
;
;               JMP     SIGN_C_D_1          ;Dummy jump
;SIGN_C_D_1:
;               DB      0DFh,0FFh           ;Sequential mode
;
                MOV     CX,NUM_PROC         ;Set the count
                CMP     CX,AX
                JBE     SIGN_C_1
                MOV     CX,AX
SIGN_C_1:
                CMP     LINDEX[BX],3        ;If octant = 0,1,2,7
                JB      SIGN_C_2            ;then cosine = +ve
                CMP     LINDEX[BX],7        ;else cosine = -ve
```

Figure A-2 (continue)

```
                JE      SIGN_C_2
                FCHS                    ;------------------
SIGN_C_2:
                ADD     BX,2            ;Set next address
                LOOP    SIGN_C_1        ;------------------
;
;               JMP     SIGN_C_D_2      ;Dummy jump
;SIGN_C_D_2:
;               DB      0DFh,0FEh       ;Scalar mode
;               JMP     SIGN_C_D_3      ;Dummy jump
;SIGN_C_D_3:
;               DB      0DFh,0FDh       ;Parallel mode
;
                POP     CX              ;Restore the
                POP     BX              ;registers
                POP     AX              ;------------------
                RET
SIGN_C          ENDP



                PUBLIC  H_TRAN
H_TRAN          PROC    NEAR
;-------------------------------------------------------;
; This procedure calculates the FHT.                    ;
; Reads : X, Y, COS, SIN, POWER, N                      ;
; Writes : X, Y                                         ;
; Intermediate variables : TEMP, K, N2, NM, T           ;
; Refer to the BASIC program for more information       ;
;-------------------------------------------------------;

                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX
                PUSH    DX
                PUSH    BP
                PUSH    SI
                PUSH    DI              ;------------------
                MOV     AX,N            ;T=N
                MOV     T,AX            ;------------------
                MOV     K,1             ;Set K = 1
                XOR     BX,BX           ;Set BX = 0
                MOV     CX,POWER        ;Set 1st loop,
TRAN_POWER:                             ;CX = POWER
                PUSH    CX              ;Save CX
                MOV     AX,N            ;Set input para. for
                CALL    Y_TO_X          ;subroutine and call
```

Figure A-2 (continue)

```
                MOV     AX,K            ;N2 = K
                MOV     N2,AX           ;------------------
                MOV     CL,2            ;2 for Single P. real
                SHL     AX,CL           ;3 for Double P. real
                MOV     N2_ADDRESS,AX   ;Convert N2 to addr
                SHL     K,1             ;K = K + K
                MOV     AX,K            ;--------------------
                SHL     AX,CL           ;Convert K to addr
                MOV     K_ADDRESS,AX    ;--------------------
                SHR     T,1             ;T=T/2
                XOR     DX,DX           ;Set DX = 0
                MOV     CX,N2           ;Set 2nd loop,
TRAN_I_LP:                              ;CX = N2
                MOV     BP,DX           ;BP = DX
                MOV     SI,N2_ADDRESS   ;SI = N2 + DX (Addr)
                ADD     SI,DX           ;-------------------
                MOV     DI,K_ADDRESS    ;DI = K - DX (Addr)
                SUB     DI,DX           ;-------------------
                MOV     AX,T            ;Set  3rd loop,
TRAN_T_LP:                              ;CX = T
                FINIT                   ;Initialize 8087
                PUSH    BX              ;Load W=X[BP]
                MOV     BX,BP           ;sequentially
                ADD     BX,X
                CALL    SEQ_LD_4
                POP     BX              ;-------------------
                FLD     COS[BX]         ;Load COS
                PUSH    BX              ;Load X[SI]
                MOV     BX,SI           ;sequentially
                ADD     BX,X
                CALL    SEQ_LD_4
                POP     BX              ;-------------------
                FLD     SIN[BX]         ;Load SIN
                PUSH    BX              ;Load X[DI]
                MOV     BX,DI           ;sequentially
                ADD     BX,X
                CALL    SEQ_LD_4
                POP     BX              ;-------------------
                FMULP   ST(1),ST        ;ST(4)=X[DI]*SIN[BX]
                FXCH    ST(2)           ;ST(4)=COS[BX]
                FMULP   ST(1),ST        ;ST(5)=X[SI]*COS[BX]
                FADDP   ST(1),ST        ;ST(6)=U
                FLD     ST(1)           ;ST(5)=W
                FADD    ST,ST(1)        ;ST(5)=W+U
                PUSH    BX              ;Store Y[BP]
                LEA     BX,Y[BP]        ;sequentially
                CALL    SEQ_ST_4
                POP     BX              ;------------------
```

Figure A-2 (continue)

```
                FSUBR   ST,ST(1)        ;ST(6)=W-U
                PUSH    BX              ;Store Y[SI]
                LEA     BX,Y[SI]        ;sequentially
                CALL    SEQ_ST_4
                POP     BX              ;-----------------
                PUSH    AX
                MOV     AX,K_ADDRESS    ;Set next addr for
                PUSH    CX              ;BP,SI,DI
                MOV     CX,NUM_PROC
                CMP     CX,AX
                JBE     TRAN_INC_1
                MOV     CX,AX
TRAN_INC_1:
                ADD     BP,AX
                ADD     SI,AX
                ADD     DI,AX
                LOOP    TRAN_INC_1      ;-----------------
                POP     CX
                POP     AX
                SUB     AX,NUM_PROC
                CMP     AX,0
                JLE     TRAN_B
                JMP     TRAN_T_LP       ;3rd loop end
TRAN_B:
                ADD     BX,4            ;Set next addr for
                ADD     DX,4            ;BX, DX
                DEC     CX
                OR      CX,CX
                JZ      TRAN_C
                JMP     TRAN_I_LP
TRAN_C:
                POP     CX
                DEC     CX              ;LOOP will cause
                OR      CX,CX           ;out of range error
                JZ      TRAN_DONE
                JMP     TRAN_POWER      ;3rd loop end
TRAN_DONE:
                MOV     AX,N            ;Set input para. for
                CALL    Y_TO_X          ;subroutine and call
                POP     DI              ;Restore the
                POP     SI              ;registers
                POP     BP
                POP     DX
                POP     CX
                POP     BX
                POP     AX              ;-----------------
                RET
H_TRAN          ENDP
```

Figure A-2 (continue)

```
.DATA

POWER           DW      0h
BASE            DW      2h
TEMP            DW      0h
X               DW      0h
Y               DD      256 DUP(0)
SIN             DD      256 DUP(0)
COS             DD      256 DUP(0)
K               DW      0h
K_ADDRESS       DW      0h
N2              DW      0h
N2_ADDRESS      DW      0h
T               DW      0h
I               DW      0h
ANGLE           DD      256 DUP(0)
LINDEX          DW      256 DUP(0)

                PUBLIC  NUM_PROC
NUM_PROC        DW      1h
                PUBLIC  N
N               DW      0h

.STACK
                END
```

Figure A-2 (continue)

```
.MODEL MEDIUM
.CODE
;-------------------------------------------------------;
; Program Name : P2                                     ;
; This subroutine (procedure) is to calculate 2 to the  ;
; power of P, where P is the input parameter.           ;
; From compiled BASIC program : CALL TWO_TO_P(P,N)      ;
;                       where N = 2 ^ P                 ;
; Author : Boon Pock Lim                                ;
; Date   : March 10, 1988                               ;
;-------------------------------------------------------;
                PUBLIC  P2
P2              PROC    FAR
                PUSH    BP              ;Save BP
                MOV     BP,SP           ;BP=SP
                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX              ;-----------------
                MOV     BX,[BP]+8       ;1st Arg.= P
                MOV     CX,WORD PTR [BX] ;CX=P
                MOV     AX,1
                SHL     AX,CL           ;AX = 2^P
                MOV     BX,[BP]+6       ;2nd Arg. = N
                MOV     WORD PTR [BX],AX
                POP     CX              ;Restore the
                POP     BX              ;registers
                POP     AX
                POP     BP              ;-----------------
                RET     4
P2              ENDP
                END
```

Figure A-3 P2 program listing

```
.8087
.MODEL MEDIUM
.CODE
;-------------------------------------------------------;
; Program Name : SCALE                                  ;
; This subroutine (procedure) is to scale the result from ;
; the FHT by N.                                         ;
; From compiled BASIC program : CALL SCALE(X(0),N)      ;
; Author : Boon Pock Lim                                ;
; Date   : March 10, 1988                               ;
;-------------------------------------------------------;
                EXTRN   SEQ_LD_4:FAR
                EXTRN   SEQ_ST_4:FAR

                PUBLIC  SCALE
SCALE           PROC    FAR
                PUSH    BP              ;Save BP
                MOV     BP,SP           ;BP=SP
                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX              ;-----------------
                MOV     BX,[BP]+6       ;2nd Arg.
                MOV     AX,WORD PTR [BX]
                MOV     N,AX
                MOV     BX,[BP]+8       ;1st Arg.
;
;               JMP     SC_D_1          ;Dummy jump
;SC_D_1:
;               DB      0DFh,0FDh       ;Parallel mode
;
                FINIT
                FILD    N
SC_RTN_1:
                CALL    SEQ_LD_4        ;Load X sequentially
                FDIV    ST,ST(1)        ;Scaling
                CALL    SEQ_ST_4        ;Store X sequentially
                MOV     CX,NUM_PROC     ;Set next address
                CMP     CX,AX
                JBE     SC_RTN_2
                MOV     CX,AX
SC_RTN_2:
                ADD     BX,4
                LOOP    SC_RTN_2        ;-----------------
                SUB     AX,NUM_PROC
                CMP     AX,0
                JG      SC_RTN_1
;
;               JMP     SC_D_2          ;Dummy jump
```

Figure A-4 SCALE program listing

```
;SC_D_2:
;               DB      0DFh,0FEh       ;Scalar mode
;
                POP     CX              ;Restore the
                POP     BX              ;registers
                POP     AX
                POP     BP              ;----------------
                RET     4
SCALE           ENDP

.DATA

                EXTRN   NUM_PROC:WORD
                EXTRN   N:WORD

                END
```

Figure A-4 (continue)

```
.8087
.MODEL MEDIUM
.CODE
;-------------------------------------------------------;
; Program Name : DFT                                    ;
; This subroutine(procedure) is to calculate the DFT    ;
; from the DHT                                          ;
; From compiled BASIC program :                         ;
;                   CALL DFT(X(0),N,RE(0),IM(0))        ;
;                   where RE : real part of DFT         ;
;                         IM : imaginary part of DFT    ;
; Author : Boon Pock Lim                                ;
; Date   : March 10, 1988                               ;
;-------------------------------------------------------;

                EXTRN   SEQ_LD_4:FAR
                EXTRN   SEQ_ST_4:FAR

                PUBLIC  DFT
DFT             PROC    FAR
                PUSH    BP              ;Save BP
                MOV     BP,SP           ;BP=SP
                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX
                PUSH    DX
                PUSH    SI
                PUSH    DI              ;------------------
                MOV     SI,[BP]+12      ;1st Arg.
                MOV     BX,[BP]+10      ;2nd Arg.
                MOV     AX,[BX]         ;N
                DEC     AX              ;N-1
                PUSH    AX
                MOV     CX,2            ;(N-1)*4 - offset
                SHL     AX,CL           ;------------------
                MOV     DI,AX           ;Last element of X
                ADD     DI,SI           ;------------------
                POP     AX
                MOV     BX,[BP]+8       ;3rd Arg. - RE
                MOV     BP,[BP]+6       ;4th Arg. - IM
                FINIT                   ;Reset 8087
                FLDZ                    ;ST(7)=0
                FLD     DWORD PTR [SI]  ;ST(6)=X(0)
                FSTP    DWORD PTR [BX]  ;RE = X(0)
                FSTP    DWORD PTR [BP]  ;IM = 0
;
;               JMP     DFT_D_1         ;Dummy jump
;DFT_D_1:
```

Figure A-5 DFT program listing

```
;               DB      0DFh,0FDh       ;Parallel mode
;               FINIT
;
                FLD1                    ;ST(7)=1
                FCHS                    ;ST(7)=-1 (/2)
                MOV     DX,4            ;Set next addresses
                ADD     SI,DX
                ADD     BX,DX
                ADD     BP,DX           ;------------------
DFT_RTN_1:
                PUSH    BX              ;Load X(k)
                MOV     BX,SI           ;sequentially
                CALL    SEQ_LD_4
                MOV     BX,DI           ;Load X(n-k)
                CALL    SEQ_LD_4        ;sequentially
                POP     BX              ;------------------
                FLD     ST(0)
                FINCSTP
                FXCH    ST(2)
                FDECSTP
                FADD    ST,ST(2)
                FSCALE
                CALL    SEQ_ST_4        ;Real #
                FXCH    ST(2)
                FSUBRP  ST(1),ST
                FSCALE
                PUSH    BX
                MOV     BX,BP           ;Imaginary #
                CALL    SEQ_ST_4
                POP     BX
                MOV     CX,NUM_PROC     ;Set next address
                CMP     CX,AX
                JBE     DFT_RTN_2
                MOV     CX,AX
DFT_RTN_2:
                SUB     DI,DX           ;Set next addresses
                ADD     SI,DX
                ADD     BX,DX
                ADD     BP,DX
                LOOP    DFT_RTN_2       ;------------------
                SUB     AX,NUM_PROC
                CMP     AX,0
                JG      DFT_RTN_1
;
;               JMP     DFT_D_2         ;Dummy jump
;DFT_D_2:
;               DB      0DFh,0FEh       ;Scalar mode
;
```

Figure A-5 (continue)

```
                POP     DI              ;Restore the
                POP     SI              ;registers
                POP     DX
                POP     CX
                POP     BX
                POP     AX
                POP     BP              ;------------------
                RET     8
DFT             ENDP


.DATA

                EXTRN   NUM_PROC:WORD

.STACK
                END
```

Figure A-5 (continue)

```
.8087
.MODEL  MEDIUM
.CODE

                PUBLIC  SEQ_LD_2
SEQ_LD_2        PROC    FAR
;-------------------------------------------------;
; This procedure loads 2-byte data sequentially.  ;
; Input parameters : AX = # of data to be processed ;
;                    BX = index of data           ;
; Reads : NUM_PROC, data                          ;
;-------------------------------------------------;
                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX
                PUSH    DX
                PUSH    SI              ;-----------------
;
;               JMP     S2_LD_D1        ;Dummy jump
;S2_LD_D1:
;               DB      0DFh,0FFh       ;Sequential mode
;
                XOR     DX,DX           ;If AX < # of
                MOV     CX,NUM_PROC     ;processors then
                MOV     SI,CX           ;CX=AX else
                CMP     CX,AX           ;CX=# or processors
                JBE     S_2_LD_1        ;DX : Flag
                MOV     CX,AX
                MOV     DX,1
S_2_LD_1:
                FILD    WORD PTR [BX]   ;Load data
                ADD     BX,2            ;sequentially
                LOOP    S_2_LD_1
                OR      DX,DX
                JZ      S_2_LD_3
                MOV     CX,SI
                SUB     CX,AX
S_2_LD_2:
                FLD1                    ;Load dummy 1 to
                LOOP    S_2_LD_2        ;other processors.
S_2_LD_3:
;
;               JMP     S2_LD_D2        ;Dummy jump
;S2_LD_D2:
;               DB      0DFh,0FEh       ;Scalar mode
;               JMP     S2_LD_D3        ;Dummy jump
;S2_LD_D3:
;               DB      0DFh,0FDh       ;Parallel mode
```

Figure A-6 SEQ program listing

```
                POP     SI              ;Restore the
                POP     DX              ;registers
                POP     CX
                POP     BX
                POP     AX              ;----------------
                RET
SEQ_LD_2        ENDP

                PUBLIC  SEQ_ST_2
SEQ_ST_2       PROC    FAR
;-------------------------------------------------------;
; This procedure stores 2-byte data sequentially.       ;
; Input parameters : AX = # of data to be processed     ;
;                    BX = index of data                 ;
; Reads : NUM_PROC                                      ;
; Writes : data                                         ;
;-------------------------------------------------------;
                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX              ;------------------
;
;               JMP     S2_ST_D1        ;Dummy jump
;S2_ST_D1:
;               DB      0DFh,0FFh       ;Sequential mode
;
                MOV     CX,NUM_PROC     ;If AX<# of
                CMP     CX,AX           ;processors then
                JBE     S_2_ST_1        ;CX = AX else
                MOV     CX,AX           ;CX=# of processors
S_2_ST_1:
                FISTP   WORD PTR [BX]   ;Store data
                ADD     BX,2            ;sequentially
                LOOP    S_2_ST_1        ;------------------
;
;               JMP     S2_ST_D2        ;Dummy jump
;S2_ST_D2:
;               DB      0DFh,0FEh       ;Scalar mode
;               JMP     S2_ST_D3        ;Dummy jump
;S2_ST_D3:
;               DB      0DFh,0FDh       ;Parallel mode
;
                POP     CX              ;Restore the
                POP     BX              ;registers
                POP     AX              ;------------------
                RET
SEQ_ST_2        ENDP

                PUBLIC  SEQ_LD_4
```

Figure A-6 (continue)

```
SEQ_LD_4        PROC    FAR
;-------------------------------------------------------;
; This procedure loads 4-byte data sequentially.        ;
; Input parameters : AX = # of data to be processed     ;
;                    BX = index of data                 ;
; Reads : NUM_PROC, data                                ;
;-------------------------------------------------------;
                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX
                PUSH    DX
                PUSH    SI              ;----------------
;
;               JMP     S4_LD_D1        ;Dummy jump
;S4_LD_D1:
;               DB      0DFh,0FFh       ;Sequential mode
;
                XOR     DX,DX           ;DX : flag
                MOV     CX,NUM_PROC     ;If AX < # of
                MOV     SI,CX           ;processors then
                CMP     CX,AX           ;CX = AX else
                JBE     S_4_LD_1        ;CX=# of processors
                MOV     CX,AX
                MOV     DX,1            ;----------------
S_4_LD_1:
                FLD     DWORD PTR [BX]  ;Load data
                ADD     BX,4            ;sequentially
                LOOP    S_4_LD_1        ;----------------
                OR      DX,DX
                JZ      S_4_LD_3
                MOV     CX,SI
                SUB     CX,AX
S_4_LD_2:
                FLD1                    ;Load dummy 1 to
                LOOP    S_4_LD_2        ;other processors
S_4_LD_3:
;
;               JMP     S4_LD_D2        ;Dummy jump
;S4_LD_D2:
;               DB      0DFh,0FEh       ;Scalar mode
;               JMP     S4_LD_D3        ;Dummy jump
;S4_LD_D3:
;               DB      0DFh,0FDh       ;Parallel mode
;
                POP     SI              ;Restore the
                POP     DX              ;registers
                POP     CX
                POP     BX
```

Figure A-6 (continue)

```
                POP     AX              ;-----------------
                RET
SEQ_LD_4        ENDP


                PUBLIC  SEQ_ST_4
SEQ_ST_4        PROC    FAR
;-------------------------------------------------;
; This procedure stores 4-byte data sequentially. ;
; Input parameters : AX = # of data to be processed ;
;                    BX = index of data           ;
; Reads : NUM_PROC                                ;
; Writes : data                                   ;
;-------------------------------------------------;
                PUSH    AX              ;Save the registers
                PUSH    BX
                PUSH    CX              ;-----------------
;
;               JMP     S4_ST_D1        ;Dummy jump
;S4_ST_D1:
;               DB      0DFh,0FFh       ;Sequential mode
;
                MOV     CX,NUM_PROC     ;If AX < # of
                CMP     CX,AX           ;processors then
                JBE     S_4_ST_1        ;CX=AX else # of
                MOV     CX,AX           ;CX=# of processors
S_4_ST_1:
                FSTP    DWORD PTR [BX]  ;Store data
                ADD     BX,4            ;sequentially
                LOOP    S_4_ST_1        ;-----------------
;
;               JMP     S4_ST_D2        ;Dummy jump
;S4_ST_D2:
;               DB      0DFh,0FEh       ;Scalar mode
;               JMP     S4_ST_D3        ;Dummy jump
;S4_ST_D3:
;               DB      0DFh,0FDh       ;Parallel mode
;
                POP     CX              ;Restore the
                POP     BX              ;registers
                POP     AX              ;-----------------
                RET
SEQ_ST_4        ENDP

.DATA
                EXTRN   NUM_PROC:WORD

                END
```

Figure A-6 (continue)