MICROPROCESSOR APPLICATION IN WALSH-FOURIER CONVERSION

MICROPROCESSOR APPLICATION IN WALSH-FOURIER CONVERSION

by

PRADEEP N. BANSOD, B.Tech. (Indian Institute of Technology, Delhi)

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Engineering

McMaster University

November 1975

MASTER OF ENGINEERING
(Electrical Engineering)

McMaster University
Hamilton, Ontario, Canada

TITLE: Microprocessor Application in Walsh-Fourier Conversion

AUTHOR: Pradeep N. Bansod, B.Tech. (Indian Institute of Technology, Delhi)

SUPERVISOR: Dr. R. Kitai, D.Sc.

NUMBER OF PAGES: (x), 104

ABSTRACT

A microprocessor based system to convert the Walsh spectrum of a frequency limited signal to its Fourier spectrum has been designed and built. The entire processing hardware is implemented on a single PLS-401 card, which consists of an Intel 4004 microprocessor, read only memories to store the conversion program and matrices of constants, random access memories to store results, and input/output ports. The converter can process up to 32 coefficients, and utilizes an 8-bit word length. For test purposes, the Walsh spectra are programmed into a read only memory, and the Fourier spectra are displayed in binary form on an LED matrix. The maximum conversion time is 1.81 seconds, and the maximum absolute error is 2.03% of the largest possible coefficient.

ACKNOWLEDGEMENTS

The author particularly wishes to thank Prof. R. Kitai for his helpful advice and guidance during the course of this work. The author would also like to thank Mr. Sanjeev Shroff for some very useful discussions, and Mr. Sorin Cohn-Sfetcu for his help in preparing this thesis.

Finally, the author would like to thank McMaster University for financial assistance received during the course of this work.

TABLE OF CONTENTS

APPENDICES

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

$\underline{a}$	Vector of cosine coefficients
$\underline{A}$	Vector of cal coefficients
$\underline{b}$	Vector of sine coefficients
$\underline{B}$	Vector of cal coefficients
CPU	Central Processing Unit
f	Index of Fourier spectral coefficient. It also represents the row number of the $\underline{F}$ matrix.
$\underline{F}$	Matrix of magnitudes of conversion elements
$\underline{F}_a$	Cal/Cosine conversion matrix
$\underline{F}_b$	Sal/Sine conversion matrix
$\underline{F}_{64}$	Matrix of unique elements in $\underline{F}$
GND	Ground
$\underline{K}$	Compensation matrix
LSB	Least significant bit
MOS	Metal oxide semiconductor
MSB	Most significant bit
Q	Row number of $\underline{F}_{64}$ matrix
RAM	Random access memory
ROM	Read only memory
s	Index of Walsh spectral coefficient. It also represents the column number of the $\underline{F}$ matrix.
TTL	Transistor transistor logic
V_{CC}	Most positive supply voltage

V_{DD}	Most negative supply voltage
x	Number of ones to the right of the least significant zero in the binary representation of f
X	Column number of the $\underline{F}_{64}$ matrix

CHAPTER I

INTRODUCTION

The microprocessor has developed over the last few years into a computer built on a few low cost chips. This has facilitated the use of software techniques in many areas where, previously, computer applications were too expensive. Thus, it is finding widespread use in digital signal processing, and this report describes a microprocessor in such an application.

Sinusoids have long been used to characterize wave phenomena. Many naturally occuring waves are sinusoidal in form, and the use of Fourier analysis is very helpful for studying linear systems, particularly in electrical engineering. However, with the continuing advances in digital electronics, it is increasingly attractive to consider alternative methods of signal analysis, especially those employing binary basis functions. The Walsh functions form such a binary orthogonal set, and have been used to build a Walsh spectral analyzer [1].

For a frequency limited signal, a Fourier spectrum is still preferred, and the Walsh spectrum needs to be converted to the corresponding Fourier spectrum. This thesis discusses the design and implementation of a microprocessor based Walsh-Fourier converter.

One method which has been proposed [2] to implement this conversion utilizes standard integrated circuits to control a Fairchild 9344 fast multiplier, and various memories for storage of constants. An

alternative method would be to perform the conversion by a software routine running on a digital computer. Until recently, the high cost of computers prevented their use in a dedicated instrument. However, in recent years low cost microprocessors have become available, and these can be used to build programmable digital instruments, such as the Walsh-Fourier converter.

The microprocessor instrument has a number of advantages over a random logic converter. It is cheaper, smaller, and consumes less power. The number of devices used is much less, reducing assembly requirements and increasing reliability. It is more flexible, allowing modifications in the functions to be easily implemented by merely changing the program. Indeed, while the system may be used as a dedicated instrument, it can also be used for many other applications, again only involving reprogramming.

The main disadvantage of the microprocessor is its low speed. An improvement can be achieved by using the microprocessor to control a fast multiplier, but again this entails higher cost and lower reliability.

Microprocessor costs are continually falling, and their performance is improving; thus their use in a Walsh-Fourier converter is indeed very attractive.

Chapter II describes the Walsh Functions and the process of Walsh to Fourier conversion.

Chapter III describes the microprocessor hardware.

Chapter IV details the conversion program.

Chapter V discusses the test results and design features, together with suggestions for further improvements.

CHAPTER II

WALSH FUNCTIONS AND WALSH-FOURIER CONVERSION

2.1 Definition

The Walsh functions are a complete set of orthonormal functions, which take on only two values, +1 and -1. A natural Walsh set comprises 2^n functions wal(k,t), $0 \leq k \leq 2^n-1$, where n is the number of bits in the binary representation of the order, k, of the Walsh function. The normalized time, t, takes values between 0 and 1.

The Walsh set for n=4 is shown in Figure 2.1. As can be seen from the figure, the functions change sign only when $t = \ell.(\frac{1}{2})^m$, where ℓ and m are integers. The sequency of any Walsh function is defined as half the number of sign changes in the unit interval, and is denoted by the letter s. As in the case of Fourier functions, the Walsh functions can be divided into two subsets: the cal functions and the sal functions. Wal(0,t), which corresponds to d.c., is generally regarded as cal(0,t). Subsequent Walsh functions are then alternately sal and cal. For every sequency value there is one cal and one sal function, and the relation between the sequency and the order is:

$$k = 2s \qquad \text{for cal functions} \qquad (2\text{-}1a)$$

$$k = 2s-1 \qquad \text{for sal functions} \qquad (2\text{-}1b)$$

A formal definition of the Walsh functions, as given by Cardot[3], is as follows:

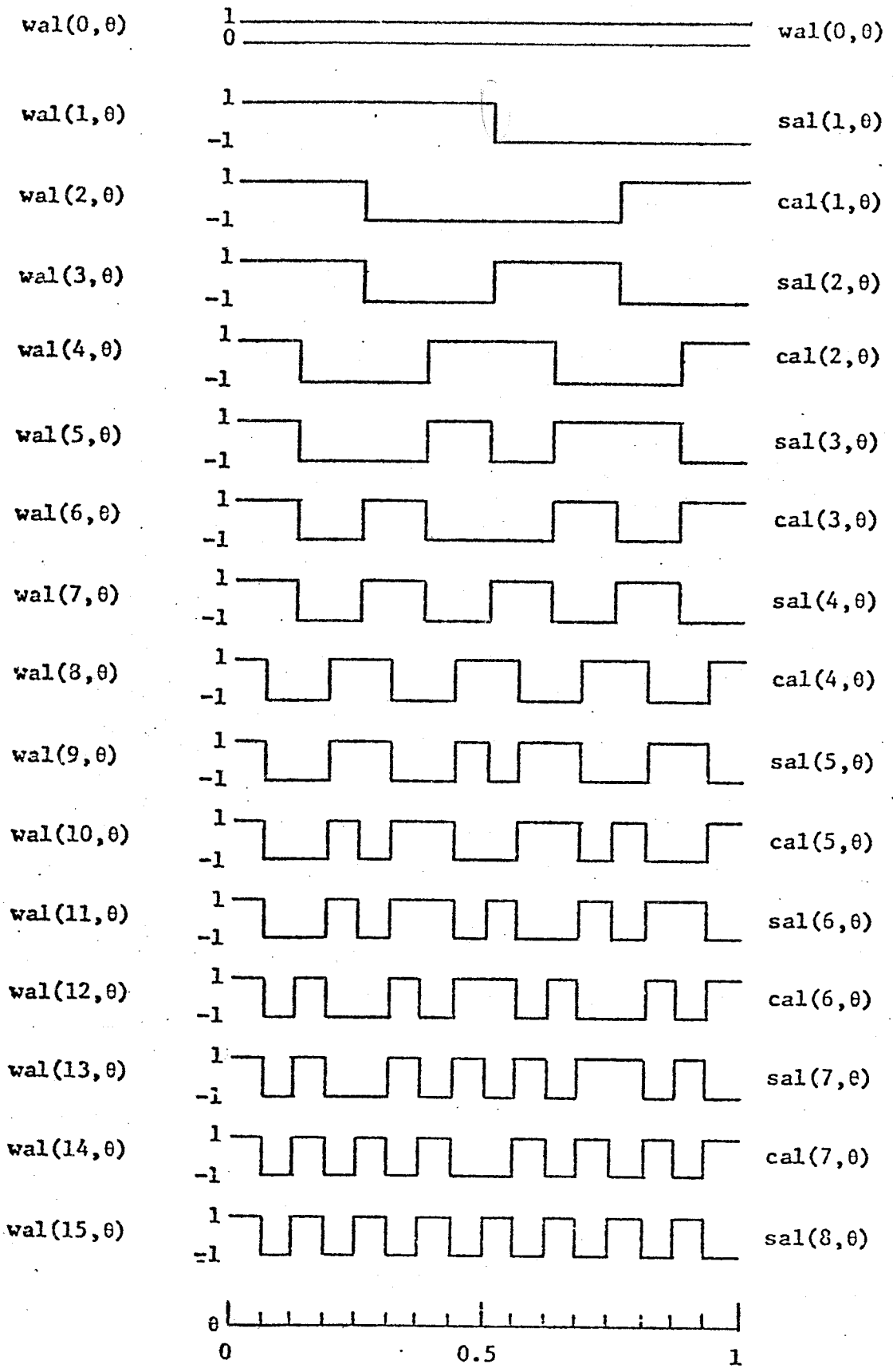


Fig. 2.1 Walsh Functions

The order, k, is expressed in the form

$$k = \sum_{r=0}^{m-1} (2^r k_r) \tag{2-2}$$

where $\{k_r\}$ are the digits, 0 or 1, of the binary representation of k, and 2^m is any power of 2 that exceeds k. Then,

$$wal(k,t) = \prod_{r=0}^{m-1} sgn(\cos^{k_r} 2^r \pi t) \tag{2-3}$$

The signum function, sgn(x), is ±1 according as x>0 or x<0

Each one of the above cosines turns into a square wave, when k_r is 1, the period depending on r. For $k_r = 0$, the cosine becomes the constant 1. The Walsh function, then, is the product of as many such square waves as there are ones in the binary representation of k.

As an example, let us take the case of k = 5. Then,

$$k_{binary} = 101$$

and $k_0 = 1$, $k_1 = 0$, $k_2 = 1$. Wal(5,t) is, therefore, the product of $sgn(\cos 2^0 \pi t) = sgn(\cos \pi t)$ and $sgn(\cos 2^2 \pi t) = sgn(\cos 4\pi t)$. Figure 2.2, which shows $\cos \pi t$, $sgn(\cos \pi t)$, $\cos 4\pi t$, $sgn(\cos 4\pi t)$ and wal(5,t), illustrates how each cosine is turned into a square wave, and how these contribute to the Walsh function.

2.2. Walsh Spectral Analysis

It is an established fact that any periodic waveform can be expressed in terms of any orthonormal set of functions, the most common of which are the Fourier functions. Fourier analysis utilizes the frequency

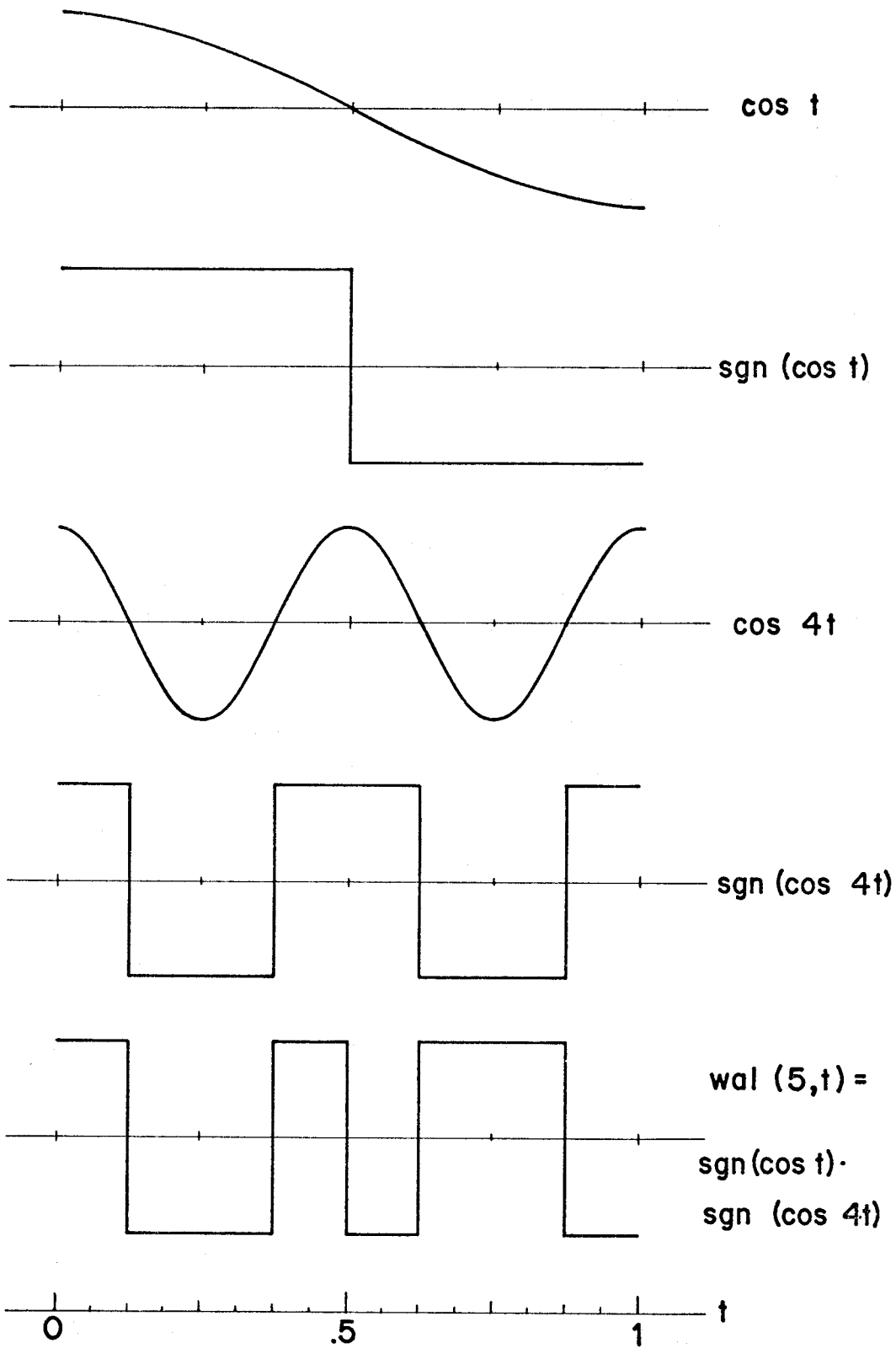


Fig. 2.2 The Composite Square Waves in Wal(5,t)

concept, which is particularly attractive in electrical engineering, as this helps to simplify network analysis. Sinusoids remain sinusoids after various mathematical operations such as addition, subtraction, integration, and differentiation. Thus, if a linear system is excited by a sum of sinusoids, the output will also be a sum of sinusoids, and it is easier to determine the response of a system this way, rather than by solving its differential equation.

Conventional frequency analyzers, however, are generally limited to frequencies above 20 Hertz. At lower frequencies, the frequency selective elements become increasingly difficult to realize. Thus, in this frequency range, a digital approach becomes necessary.

As seen in the previous section, Walsh functions also form an orthonormal set, which may be used for spectral anlysis. A function, f(t), may be expanded in a Walsh series in the form

$$f(t) = A_0 + \sum_{s=1}^{\infty} [A_s cal(s,t) + B_s sal(s,t)] \qquad (2\text{-}4)$$

where s is the sequency of the cal or sal function. The Walsh coefficients are obtained from the relations:

$$A_0 = \frac{1}{T}\int_0^T f(t).dt \qquad (2\text{-}5)$$

$$A_s = \frac{1}{T}\int_0^T f(t).cal(s,t).dt \qquad (2\text{-}6)$$

$$B_s = \frac{1}{T}\int_0^T f(t).sal(s,t).dt \qquad (2\text{-}7)$$

T is the period of f(t).

The Walsh functions are bipolar in nature, and readily lend themselves for use in a binary digital system. A digital Walsh spectral analyzer, in evaluating the above integrals, would only have to accumulate the sampled values of f(t), with the sign depending on the sign of the sample and that of the Walsh function. The multiplications which are required in Fourier analysis are eliminated. This simplifies the hardware and increases the speed of the analyzer. Even in the case of computer analysis, a fast Walsh transform is much faster than a fast Fourier transform. A digital Walsh spectral analyzer, to generate the first 64 coefficients of a periodic or random signal, has been designed and built by Siemens[1].

2.3 Walsh-Fourier Conversion

While the Walsh spectrum of a periodic waveform is easily computed, in most practical applications we are more concerned with the Fourier spectrum. Thus, it is necessary to convert the Walsh spectrum to the corresponding Fourier spectrum, using the process derived by Siemens and Kitai [4].

2.3.1 Conversion Equations

Each of the cal or sal functions in equation (2-4) can be expanded in a Fourier series, with the series for cal containing only cosine terms, and that for sal containing only sines. The coefficients of the series are found by evaluating the two expressions:

$$a_{f,s} = 2\int_0^1 \mathrm{cal}(s,t).\cos 2\pi ft.dt \qquad (2\text{-}8)$$

$$b_{f,s} = 2\int_0^1 sal(s,t).\sin 2\pi ft.dt \qquad (2\text{-}9)$$

where $a_{f,s}$ and $b_{f,s}$ are the f^{th} cosine and sine coefficients of cal(s,t) and sal(s,t), respectively.

The Fourier series for each Walsh function is now substituted back into equation (2-4). Coefficients of like frequency components are collected, and equated to the corresponding terms in the Fourier expansion of f(t):

$$f(t) = \frac{a_0}{2} + \sum_{f=1}^{\infty} [a_f.\cos 2\pi ft + b_f.\sin 2\pi ft] \qquad (2\text{-}10)$$

Thus, we obtain the Walsh-Fourier conversion equations:

$$a_f = \sum_{s=1}^{\infty} a_{f,s}.A_s \qquad (2\text{-}11)$$

$$b_f = \sum_{s=1}^{\infty} b_{f,s}.B_s \qquad (2\text{-}12)$$

$$a_0 = 2.A_0 \qquad (2\text{-}13)$$

In effect, this is equivalent to multiplying the vectors of Walsh coefficients, $\underline{A}$ and $\underline{B}$, by conversion matrices $\underline{F}_a$ and $\underline{F}_b$. The matrix form of the conversion equations is:

$$\underline{a} = \underline{F}_a.\underline{A} \qquad (2\text{-}14)$$

$$\underline{b} = \underline{F}_b.\underline{B} \qquad (2\text{-}15)$$

where $\underline{a}$ and $\underline{b}$ are the vectors of Fourier coefficients.

The first 4x4 elements of $\underline{F}_a$ and $\underline{F}_b$ are listed below:

$$\underline{F}_a = \begin{bmatrix} 4/\pi & 0 & 0.527 & 0 & \dots \\ 0 & 4/\pi & 0 & 0 & \dots \\ -4/3\pi & 0 & 1.025 & 0 & \dots \\ 0 & 0 & 0 & 4/\pi & \dots \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \end{bmatrix} \qquad (2\text{-}16)$$

$$\underline{F}_b = \begin{bmatrix} 4/\pi & 0 & -0.527 & 0 & \dots \\ 0 & 4/\pi & 0 & 0 & \dots \\ 4/3\pi & 0 & 1.025 & 0 & \dots \\ 0 & 0 & 0 & 4/\pi & \dots \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \end{bmatrix} \qquad (2\text{-}17)$$

The s^{th} column of each matrix represents the Fourier expansion of cal(s,t) or sal(s,t), with the f^{th} element in that column representing the f^{th} Fourier coefficient. For example, the first column of $\underline{F}_a$ represents the Fourier series for cal(1,t):

$$\text{cal}(1,t) = (4/\pi).\cos t - (4/3\pi).\cos 3t + \dots$$

Similarly, the rows of each matrix represent the Walsh expansion of the corresponding Fourier function. For example, the third row of $\underline{F}_b$ gives the Walsh expansion of sin3t:

$$\sin 3t = (4/3\pi).\text{sal}(1,t) + 1.025\ \text{sal}(3,t)$$

2.3.2 Compensation for Truncated Spectra [4]

Any Walsh spectral analyzer can only provide a limited number of Walsh coefficients, and evaluating equations (2-14) and (2-15) using these truncated spectra would introduce errors in the conversion.

However, if the signal being analyzed is frequency limited, so that the highest component is of frequency F, then these F Fourier coefficients can be exactly calculated from the first S Walsh coefficients, provided $S \geq F$, and $S = 2^{n-1}$, n being an integer. The conversion equations (2-14) and (2-15) are modified to:

$$\underline{a} = \underline{K}.\underline{F_a}.\underline{A} \qquad (2\text{-}18)$$

$$\underline{b} = \underline{K}.\underline{F_b}.\underline{B} \qquad (2\text{-}19)$$

where $\underline{K}$ is a diagonal matrix, of order SxS, with the diagonal elements given by

$$K_{f,f} = \text{sinc}^{-2}(f/2^n) \quad , \text{ for } f < 2^{n-1} \qquad (2\text{-}20a)$$

$$K_{f,f} = \frac{\text{sinc}^{-2}(1/2)}{2} \quad , \text{ for } f = 2^{n-1} \qquad (2\text{-}20b)$$

As $n \to \infty$, $K_{f,f}$ approaches 1, for all values of f.

2.3.3 An Example

Consider the frequency limited function:

$$f(t) = \cos t + 0.25\sin t - 1.25\sin 2t + \sin 3t$$

The first sixteen cal and sal coefficients of the Walsh spectrum of this function are given in table 2.1. Using this truncated spectrum, the matrix

TABLE 2.1 Walsh Spectrum of $f(t) = \cos t + 0.25\sin t - 1.25\sin 2t + \sin 3t$

Cal Coefficients

Coefficient No.	Decimal	Binary	Hexadecimal ($x2^7$)
0	0.63662	0.1010001	5 1
1	0.00000	0.0000000	0 0
2	0.26370	0.0100010	2 2
3	0.00000	0.0000000	0 0
4	-0.05245	-0.0000111	-0 7
5	0.00000	0.0000000	0 0
6	0.12663	0.0010000	1 0
7	0.00000	0.0000000	0 0
8	-0.01247	-0.0000010	-0 2
9	0.00000	0.0000000	0 0
10	-0.00517	-0.0000001	-0 1
11	0.00000	0.0000000	0 0
12	-0.02597	-0.0000011	-0 3
13	0.00000	0.0000000	0 0
14	0.06270	0.0001000	0 8
15	0.00000	0.0000000	0 0

Sal Coefficients

Coefficient No.	Decimal	Binary	Hexadecimal ($x2^7$)
0	0.37136	0.0110000	3 0
1	-0.79577	-0.1100110	-6 6
2	0.44639	0.0111001	3 9
3	0.00000	0.0000000	0 0
4	-0.35543	-0.0101101	-2 D
5	0.32962	0.0101010	2 A
6	0.11013	0.0001110	0 E
7	0.00000	0.0000000	0 0
8	-0.04613	-0.0000110	-0 6
9	0.06557	0.0001000	0 8
10	-0.10255	-0.0001101	-0 D
11	0.00000	0.0000000	0 0
12	-0.16190	-0.0010101	-1 5
13	0.15829	0.0010100	1 4
14	0.04870	0.0000110	0 6
15	0.00000	0.0000000	0 0

multiplications shown below are executed, to obtain the uncompensated Fourier spectrum. It can be seen that the values of the Fourier coefficients are substantially in error.

Cosine evaluation

$$\begin{bmatrix} 1.273 & 0 & 0.527 & \cdots & 0 \\ 0 & 1.273 & 0 & \cdots & 0 \\ -0.424 & 0 & 1.025 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1.273 \end{bmatrix} \begin{bmatrix} 0.636 \\ 0 \\ 0.264 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0.997 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Sine evaluation

$$\begin{bmatrix} 1.273 & 0 & -0.527 & \cdots & 0 \\ 0 & 1.273 & 0 & \cdots & 0 \\ 0.424 & 0 & 1.025 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1.273 \end{bmatrix} \begin{bmatrix} 0.371 \\ -0.796 \\ 0.446 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0.249 \\ -1.234 \\ 0.972 \\ \vdots \\ 0 \end{bmatrix}$$

Fig. 2.3 Evaluation of the Fourier Spectrum

To compensate for the truncation, the vector of Fourier coefficients is premultiplied by the K matrix, as shown in Fig. 2.4. The original Fourier spectrum is now obtained.

This example was also used to test the microprocessor program.

Compensation of cosine coefficients

$$\begin{bmatrix} 1.003 & 0 & 0 & \cdots & 0 \\ 0 & 1.013 & 0 & \cdots & 0 \\ 0 & 0 & 1.029 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1.234 \end{bmatrix} \begin{bmatrix} 0.997 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 1.00 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Compensation of sine coefficients

$$\begin{bmatrix} 1.003 & 0 & 0 & \cdots & 0 \\ 0 & 1.013 & 0 & \cdots & 0 \\ 0 & 0 & 1.029 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & & 1.234 \end{bmatrix} \begin{bmatrix} 0.249 \\ -1.234 \\ 0.972 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0.25 \\ -1.25 \\ 1.00 \\ \vdots \\ 0 \end{bmatrix}$$

Fig. 2.4 Compensation for Truncation

2.4 Storage of Matrices of Constants

The three matrices involved in the conversion are the $\underline{F}_a$, $\underline{F}_b$ and $\underline{K}$ matrices. Since the 4004 is a 4-bit microprocessor, it directly suggests the processing of $2^4 = 16$ coefficients. Thus, the dimension of each of these matrices is chosen to be 16x16, so each contains 256 elements. However, we shall see that, owing to certain properties of the matrices, all elements need not be stored, thus effecting saving in memory requirements. The actual scheme for storing the matrices will be detailed in Chapter IV.

In the following sections, we shall see the properties which enable saving in the memory requirements.

2.4.1 Storage of the $\underline{F}_a$ and $\underline{F}_b$ Matrices

2.4.1.1 Unique Elements in $\underline{F}_a$ and $\underline{F}_b$

The Fourier series for each Walsh function has been calculated by Siemens [1] , using a non-recursive equation for the Fourier transform of a Walsh function. He has shown that each element of the $\underline{F}_a$ matrix has the same absolute value as the corresponding element in the $\underline{F}_b$ matrix. Thus, only one matrix of absolute values need be stored, together with the separate signs for $a_{f,s}$ and $b_{f,s}$. The matrix of absolute values is denoted $\underline{F}$.

Further savings in memory space are effected by realizing that only some of the elements of $\underline{F}$ are non-zero, and these are the only ones which need to be stored. The pattern of non-zero elements is shown in Fig. 2.5.

From Fig. 2.5, it can also be seen that not all the non-zero elements are unique. Thus, only alternate rows contain unique elements. There are eight such rows, and each row has eight elements, giving a total of 64 unique elements. This matrix, which represents the only elements which need to be stored, is denoted the $\underline{F}_{64}$ matrix, and is shown in Fig. 2.6.

The magnitudes and signs of the elements of the $\underline{F}_{64}$ matrix are listed in table 2.3.

2.4.1.2 Algorithm for the Retrieval of Elements from the $\underline{F}_{64}$ Matrix

While computing the f^{th} cosine or sine coefficient, it is necessary to multiply the corresponding elements of the f^{th} row of the $\underline{F}_a$ or $\underline{F}_b$

COLUMN (s)

ROW (f)		(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)
		0	2	4	6	8	10	12	14
		1	3	5	7	9	11	13	15
(A)	0	AA	AB	AC	AD	AE	AF	AG	AH
	1	AA		AB		AC		AD	
(B)	2	BA	BB	BC	BD	BE	BF	BG	BH
	3		AA				AB		
(C)	4	CA	CB	CC	CD	CE	CF	CG	CH
	5	BA		BB		BC		BD	
(D)	6	DA	DB	DC	DD	DE	DF	DG	DH
	7				AA				
(E)	8	EA	EB	EC	ED	EE	EF	EG	EH
	9	CA		CB		CC		CD	
(F)	10	FA	FB	FC	FD	FE	FF	FG	FH
	11		BA				BB		
(G)	12	GA	GB	GC	GD	GE	GF	GG	GH
	13	DA		DB		DC		DD	
(H)	14	HA	HB	HC	HD	HE	HF	HG	HH
	15								AA

Fig. 2.5 Pattern of Non-Zero Elements in F Matrix

COLUMN (X)

ROW (Q)	0	1	2	3	4	5	6	7
0	AA	AB	AC	AD	AE	AF	AG	AH
1	BA	BB	BC	BD	BE	BF	BG	BH
2	CA	CB	CC	CD	CE	CF	CG	CH
3	DA	DB	DC	DD	DE	DF	DG	DH
4	EA	EB	EC	ED	EE	EF	EG	EH
5	FA	FB	FC	FD	FE	FF	FG	FH
6	GA	GB	GC	GD	GE	GF	GG	GH
7	HA	HB	HC	HD	HE	HF	HG	HH

Fig. 2.6 F_{64} Matrix (8x8)

TABLE 2.2 Elements of $\underline{F}_{64}$ Matrix

Word	f,s	Magnitude			Sign	
		Decimal	Binary	Hexadecimal	$a_{f,s}$	$b_{f,s}$
0	0, 0	1.2732395	1.0100011	A 3	0	0
1	0, 2	0.5273931	0.1000100	4 4	0	1
2	0, 4	0.1049050	0.0001101	0 D	1	1
3	0, 6	0.2532631	0.0100000	2 0	0	1
4	0, 8	0.0249442	0.0000011	0 3	1	1
5	0,10	0.0103322	0.0000001	0 1	1	0
6	0,12	0.0519437	0.0000111	0 7	1	1
7	0,14	0.1254031	0.0010000	1 0	0	1
8	2, 0	0.4244132	0.0110110	3 6	1	0
9	2, 2	1.0246241	1.0000011	8 3	0	0
10	2, 4	0.6846319	0.1011000	5 8	0	1
11	2, 6	0.2835838	0.0100100	2 4	0	0
12	2, 8	0.0860242	0.0001011	0 B	0	1
13	2,10	0.2076808	0.0011011	1 B	1	1
14	2,12	0.3108163	0.0101000	2 8	0	1
15	2,14	0.1287443	0.0010000	1 0	0	0
16	4, 0	0.2546479	0.0100001	2 1	0	0
17	4, 2	0.6147744	0.1001111	4 F	1	0
18	4, 4	0.9200750	0.1110110	7 6	0	0
19	4, 6	0.3811075	0.0110001	3 1	0	1
20	4, 8	0.2037062	0.0011010	1 A	1	1
21	4,10	0.4917903	0.0111111	3 F	0	1
22	4,12	0.3286038	0.0101010	2 A	0	0
23	4,14	0.1361121	0.0010001	1 1	0	1
24	6, 0	0.1818914	0.0010111	1 7	1	0
25	6, 2	0.0753419	0.0001010	0 A	1	1
26	6, 4	0.3787692	0.0110000	3 0	1	0
27	6, 6	0.9144296	0.1110101	7 5	0	0
28	6, 8	0.7504530	0.1100000	6 0	0	1
29	6,10	0.3108478	0.0101000	2 8	0	0
30	6,12	0.0618315	0.0001000	0 8	1	0
31	6,14	0.1492744	0.0010011	1 3	0	0

0 = plus sign
1 = minus sign

TABLE 2.2 (cont'd) Elements of $\underline{F}_{64}$ Matrix

Word	f,s	Magnitude			Sign	
		Decimal	Binary	Hexadecimal	$a_{f,s}$	$b_{f,s}$
32	8, 0	0.1414711	0.0010010	1 2	0	0
33	8, 2	0.0585992	0.0001000	0 8	0	1
34	8, 4	0.2945982	0.0100110	2 6	0	0
35	8, 6	0.7112230	0.1011011	5 B	1	0
36	8, 8	0.8666278	0.1101111	6 F	0	0
37	8,10	0.3589690	0.0101110	2 E	0	1
38	8,12	0.0714034	0.0001001	0 9	1	1
39	8,14	0.1723830	0.0010110	1 6	0	1
40	10, 0	0.1157490	0.0001111	0 F	1	0
41	10, 2	0.2794429	0.0100100	2 4	0	0
42	10, 4	0.4182159	0.0110110	3 6	1	0
43	10, 6	0.1732307	0.0010110	1 6	1	1
44	10, 8	0.3240918	0.0101001	2 9	1	0
45	10,10	0.7824269	0.1100100	6 4	0	0
46	10,12	0.5228009	0.1000011	4 3	0	1
47	10,14	0.2165512	0.0011100	1 C	0	0
48	12, 0	0.0979415	0.0001101	0 D	0	0
49	12, 2	0.2364517	0.0011110	1 E	1	0
50	12, 4	0.1579920	0.0010100	1 4	1	1
51	12, 6	0.0654424	0.0001000	0 8	1	0
52	12, 8	0.2157347	0.0011100	1 C	0	0
53	12,10	0.5208298	0.1000011	4 3	1	0
54	12,12	0.7794768	0.1100100	6 4	0	0
55	12,14	0.3228699	0.0101001	2 9	0	1
56	14, 0	0.0848826	0.0001011	0 B	1	0
57	14, 2	0.0351595	0.0000101	0 5	1	1
58	14, 4	0.0069937	0.0000001	0 1	0	1
59	14, 6	0.0168842	0.0000010	0 2	1	1
60	14, 8	0.1714282	0.0010110	1 6	1	0
61	14,10	0.0710079	0.0001001	0 9	1	1
62	14,12	0.3569808	0.0101110	2 E	1	0
63	14,14	0.8618279	0.1101110	6 E	0	0

0 = plus sign

1 = minus sign

by the Walsh coefficient vector, and to accumulate the products. Since it is the $\underline{F}_{64}$ matrix which is stored, and not the $\underline{F}$ matrix, it is necessary to find a relationship between the elements of $\underline{F}_{64}$ and those in $\underline{F}$. The algorithm for this has been given by Siemens [1], and is explained below.

The rows and columns of the $\underline{F}$ matrix are numbered from 0 to 15, and denoted f and s respectively. The rows and columns of the $\underline{F}_{64}$ matrix are numbered from 0 to 7, and are denoted Q and X respectively.

When computing the f^{th} Fourier coefficient, we will take the f^{th} row of $\underline{F}$, and multiply the s^{th} element in that row by the s^{th} element in the Walsh coefficient vector. Therefore, we have to find the row, Q, in $\underline{F}_{64}$ corresponding to the f^{th} row in $\underline{F}$, and the relationship between the two is:

$$f = 2^x.2Q + (2^x - 1) \qquad (2\text{-}21)$$

where x represents the number of ones to the right of the least significant zero in the binary representation of f. Since f needs 4 bits to represent it in binary, x can take values from 0 to 4.

For example, if f = 11, then f_{binary} = 1011. Hence, x = 2, Q = 1. This can be verified by referring to Figs. 2.5 and 2.6, where it can be seen that row 1 of $\underline{F}_{64}$ contains the same elements as row 11 of $\underline{F}$

Having located the correct row in $\underline{F}_{64}$, we take each element in that row and multiply it by the corresponding element in the Walsh vector. Since consecutive elements in an $\underline{F}_{64}$ row do not represent consecutive elements in an $\underline{F}$ row, we do not multiply by consecutive Walsh coefficients. Rather, the Walsh coefficient which is multiplied by the X^{th} element of

of the Qth row of $\underline{F}_{64}$ is:

$$s = 2^x.2X + (2^x - 1) \qquad (2\text{-}22)$$

where x has been computed, for a given f, from equation (2-21).

Taking the previous example of f = 11, and x = 2, we get, for successive values of X (i.e. X = 0, 1, 2, . . .)

s = 3, 11, . . .

This, again, can be verified by referring to Figs. 2.5 and 2.6, where it is seen that the first two elements of row 1 of $\underline{F}_{64}$ are the same as elements 3 and 11 of row 11 of $\underline{F}$.

The way this algorithm is actually implemented on the microprocessor is discussed in Chapter IV.

2.4.2 Storage of the K Matrix

The $\underline{K}$ matrix, though having a dimension of 16x16, is a diagonal matrix, and only the 16 diagonal elements need to be stored. These elements as calculated from equation 2-20, for n = 5, are tabulated in table 2.3.

To save the time involved in multiplying by the $\underline{K}$ matrix, we could premultiply the $\underline{F}$ and $\underline{K}$ matrices, and store the resulting matrix. However, the $\underline{F}$ matrix would lose its property of having only 64 unique elements. Since memory storage space is more critical than speed, it is preferable to store the individual matrices. Furthermore, multiplication by $\underline{K}$ really consists only of multiplying each Fourier coefficient by the corresponding diagonal element of $\underline{K}$. Thus only 16 extra multiplications are involved, and the loss of speed is not very great.

TABLE 2.3 The K Matrix

Word	Magnitude		
	Decimal	Binary	Hexadecimal
0	1.0032190	01.000000	4 0
1	1.0129507	01.000001	4 1
2	1.0294235	01.000010	4 2
3	1.0530293	01.000011	4 3
4	1.0843429	01.000101	4 5
5	1.1241502	01.001000	4 8
6	1.1734882	01.001011	4 B
7	1.2337006	01.001111	4 F
8	1.3065140	01.010100	5 4
9	1.3941420	01.011001	5 9
10	1.4994277	01.100000	6 0
11	1.6260414	o1.101000	6 8
12	1.7787576	01.110010	7 2
13	1.9638485	01.111110	7 E
14	2.1896510	10.001100	8 C
15	1.2337006	01.001111	4 F

CHAPTER III

THE MICROPROCESSOR SYSTEM

3.1 Introduction

Within the last five years, the steady advances in semiconductor technology (particularly MOS) has led to the integration of over 14,000 transistors on a single chip. This substantial increase in chip density has enabled the realization of a computer central processing unit (CPU) on a small number of LSI chips, generally less than five. This miniaturized CPU is known, owing to its small size, as a microprocessor.

Different types of microprocessors, with varying degrees of complexity, are available on the market. Some are built on a single chip (e.g. Intel's 4004), while others, such as Fairchild's PPS-25, have their functions split among several chips. However, all microprocessors perform the basic functions necessary for executing the operations and processing the data as specified by the user's program. Thus, any microprocessor must have a timing and control unit, an instruction decode and execute unit, an arithmetic unit, and some general purpose registers. Depending on the architecture of the processor, these functions may be integrated on a single chip, or distributed among the various chips forming the microprocessor.

3.2 Selection of the Microprocessor

The microprocessor which was selected for use in the converter is the Intel 4004. The major factor in selecting this microprocessor is that it was available, together with memory and input/output ports, on a

single card. This card, the PLS-401, manufactured by Pro-Log Corporation, Monterey, California, is a self-contained microprocessor system, requiring very little external circuitry. The availability of such an assembled and tested card greatly simplifies the implementation of the conversion system. The PLS-401 is described in section 3.4. Other factors which make the Intel 4004 particularly suitable for this application are described below.

The major part of the Walsh-Fourier conversion is the matrix multiplication. Thus, the microprocessor should be suitable for executing a large number of arithmetic operations. Therefore, it is important to have a number of index, or "scratch pad", registers where the operands, intermediate results and final products can be stored, and where counters can be set up. If such registers are not available, then the main random access memory (RAM) has to be used. Referencing the main memory requires many programming steps, and is slow and inefficient. The 4004 has 16 4-bit registers, which, according to the comparison given by Torrero [5], is the largest among available microprocessors[1]. As will be seen in Chapter IV, these index registers are very helpful in programming.

A "TEST" input is available on the 4004, and this can be used to allow external events to control the program execution. Since input/output is not a major consideration, an interrupt is not necessary.

The 4004 is provided with a three level hardware stack. Thus, as with index registers, the main memory need not be accessed during

---

1. The Intel 4040 microprocessor, which is not included in Torrero's comparison, does have a larger number of index registers.

subroutine calls, resulting in program simplicity and saving in execution time. Further, no software is needed to manage the stack.

The 4004 can, through the interfacing chips, be used with standard read-only memory (ROM) chips. This is particularly important, because often the cost of the memory is the largest component of the total system cost.

Finally, the 4004 is one of the cheapest microprocessors available on the market.

One disadvantage of the 4004 is that it is a 4-bit processor. As a compromise between accuracy and cost, it was decided to work with 8-bit coefficients. This implies the use of multiple precision arithmetic, which slows down the conversion. Nevertheless, the entire conversion is executed in less than two seconds, which is quite acceptable in this application.

3.3 Architecture of the Intel 4004

The 4004 is available in a 16-pin dual in line (DIP) package. The pin configuration and a description of the pin functions is given in Appendix I.

The instruction set of the 4004 is listed in Appendix II.

The block diagram of the 4004 (Fig. 3.1) consists of the following functional blocks [6] :

a). Address Register and Incrementer

The address register is a dynamic RAM cell array of 4x12 bits. It contains one level used to store the instruction address (program counter) and three levels used as a stack for subroutine calls. The address incrementer is 4-bit carry look ahead circuit which increments the address

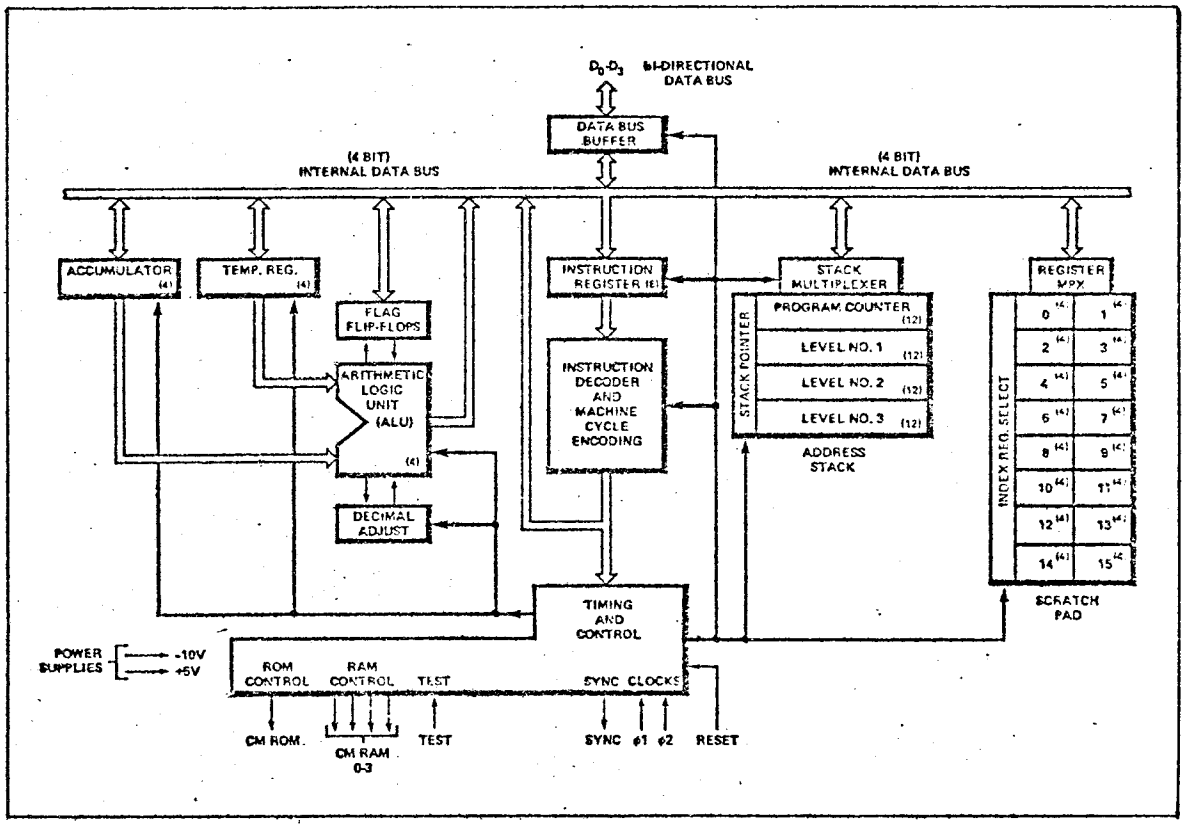


Fig. 3.1 Block Diagram of the Intel 4004 Chip

after each instruction. The contents of the stack are multiplexed onto the 4-bit internal bus.

b). Adder and Accumulator

The 4-bit adder is of the ripple-through carry type. One term of addition comes from a buffer, while the other term comes from the accumulator and carry flip-flop. The output of the adder is transferred to the accumulator and carry flip-flop. The accumulator is provided with a shifter to implement rotate right and rotate left commands. The accumulator also communicates with special ROMs which perform a code conversion to implement the DAA and KBP instructions, and with condition logic which is used in implementing the ISZ and JCN instructions.

c). Index Register

The index register is a dynamic RAM cell array of 16x4 bits and has two modes of operation. In one mode of operation the index register provides 16 directly addressable storage locations for intermediate computation and control. In the second mode, the index register provides 8 pairs of addressable storage locations for addressing RAM and ROM, as well as for storing data fetched from ROM. The index register, too, is multiplexed onto the internal bus.

d). Instruction Register, Decoder and Control

The instruction register is loaded with the contents of the internal bus through a multiplexer and holds the instruction fetched from ROM. The instructions are decoded in the instruction decoder and appropriately gated with timing signals to provide the control signals for the various functional blocks.

3.4 The PLS-401 Card

The PLS-401 system incorporates the Intel 4004 CPU, together with other necessary system components, such as ROM and RAM, and input/output ports, on a single card. A block diagram of the card is shown in Fig. 3.2, while a detailed schematic is given in Fig. 3.3.

The RAMs are Intel 4002-1 and 4002-2, and they interface directly to the CPU. Each RAM is provided with one 4-bit output port. Only one of these ports, however, has been wired out, owing to pin limitations.

Upto four ROMs, each containing 256 8-bit words, can be plugged into the sockets provided on the card. The ROMs used are Intel's 1702A. These do not interface directly with the CPU, and hence the two interface chips, 4008 and 4009, are used. In addition, a 74155 two line to four line decoder is used to select the designated ROM from a 2-bit address.

74175 latches are used as output ports, while 8234 buffers serve as input ports. Again, the 4008 and 4009 chips are used for interfacing, and the 74155 is used for selecting the port.

The clock is an astable multivibrator. The output at each of the collectors is taken to provide the two phases. The levels at this stage are +5 and 0 volts, and therefore a level shifting circuit is used to obtain the required levels of +5 and -10 volts. The clock operates at a frequency of 750 KHz.

The reset circuit, similarly, has a level shifter to obtain levels of +5 and -10 volts.

The card plugs into a 56 pin edge connector, to which all external connections can be made.

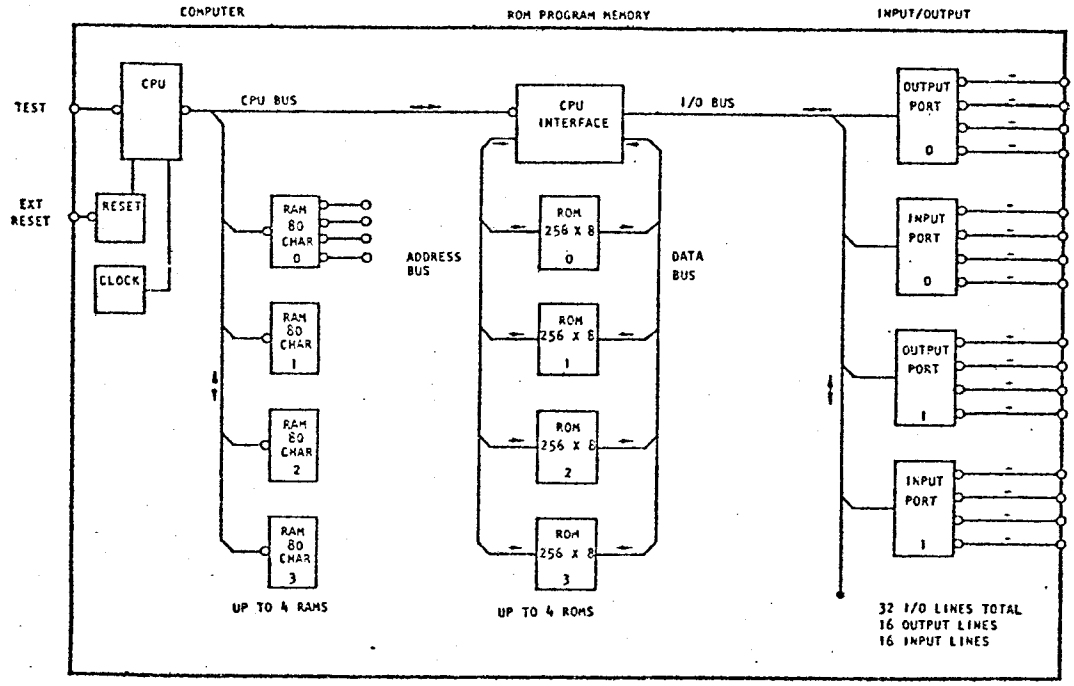


Fig. 3.2 Block Diagram of the PLS-401 Card

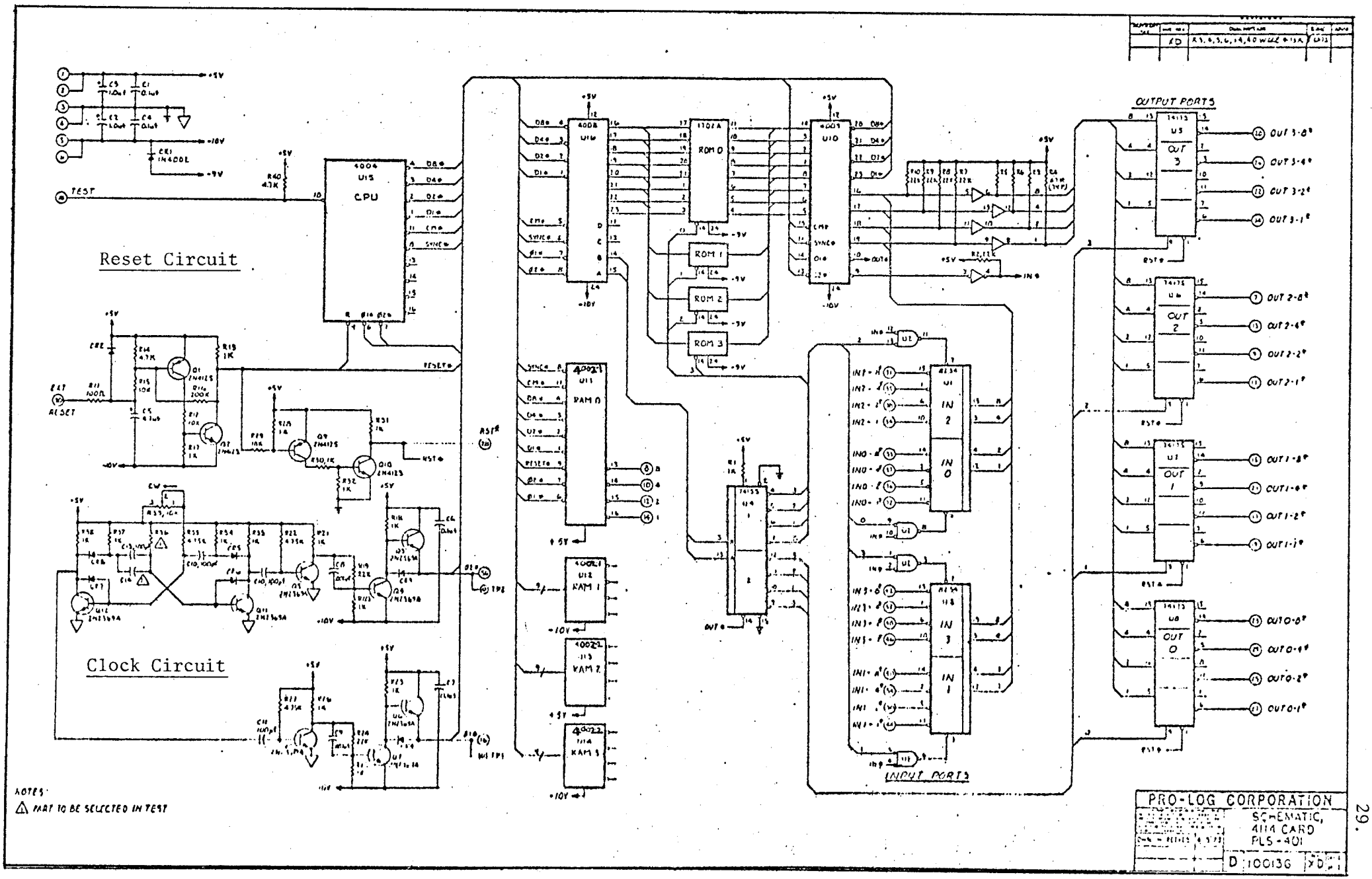


Fig. 3.3 Schematic of the PLS-401 Card

Power supplies required for the card are +5 and -10 volts. The power supply is mounted externally, and is connected to the card through the connector.

Except for the "TEST" input, all inputs and outputs have logic levels of "1" = 0 volts and "0" = +5 volts. The "TEST" input has logic levels of "1" = -10 volts and "0" = +5 volts.

All input ports are TTL compatible. The RAM output port is a MOS port, and requires a 12K pull down resistor to -10 volts for TTL compatibility. The other output ports are TTL compatible.

3.5 Display, Reset and Test Circuits

To utilize the PLS-401, some amount of external circuitry is required. This is described below.

3.5.1 Display and Input/Output

To display the states of the output ports, an LED matrix was added on a separate card. Each bit of the output ports is connected to an LED. If the bit is "1", the corresponding LED lights up. If the bit is "0", the LED is off. Since the output ports do not have sufficient current sinking capacity, 7407 buffers are used. The circuit is shown in Fig. 3.4 and the layout is shown in Fig. 3.5. The 12K pull down resistor is required only for the RAM output port.

The output ports are also brought to an output connector, through which other circuits can be connected to the microprocessor. Similarly, the input ports and the "TEST" input are brought out to an input connector. The power supplies are also available on the connectors. The scheme of input and output connections is shown in Fig. 3.6.

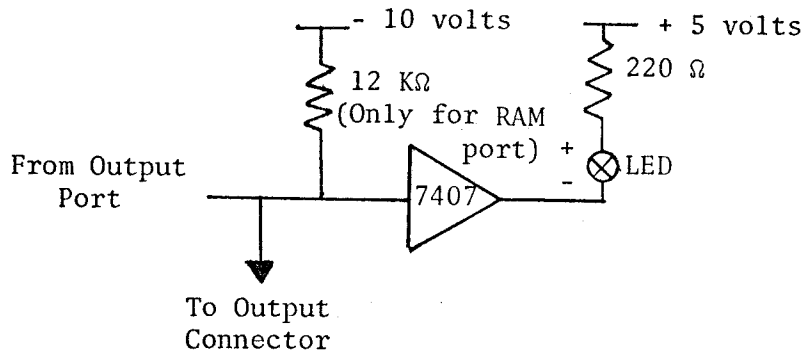


Fig. 3.4 The LED Buffer Circuit

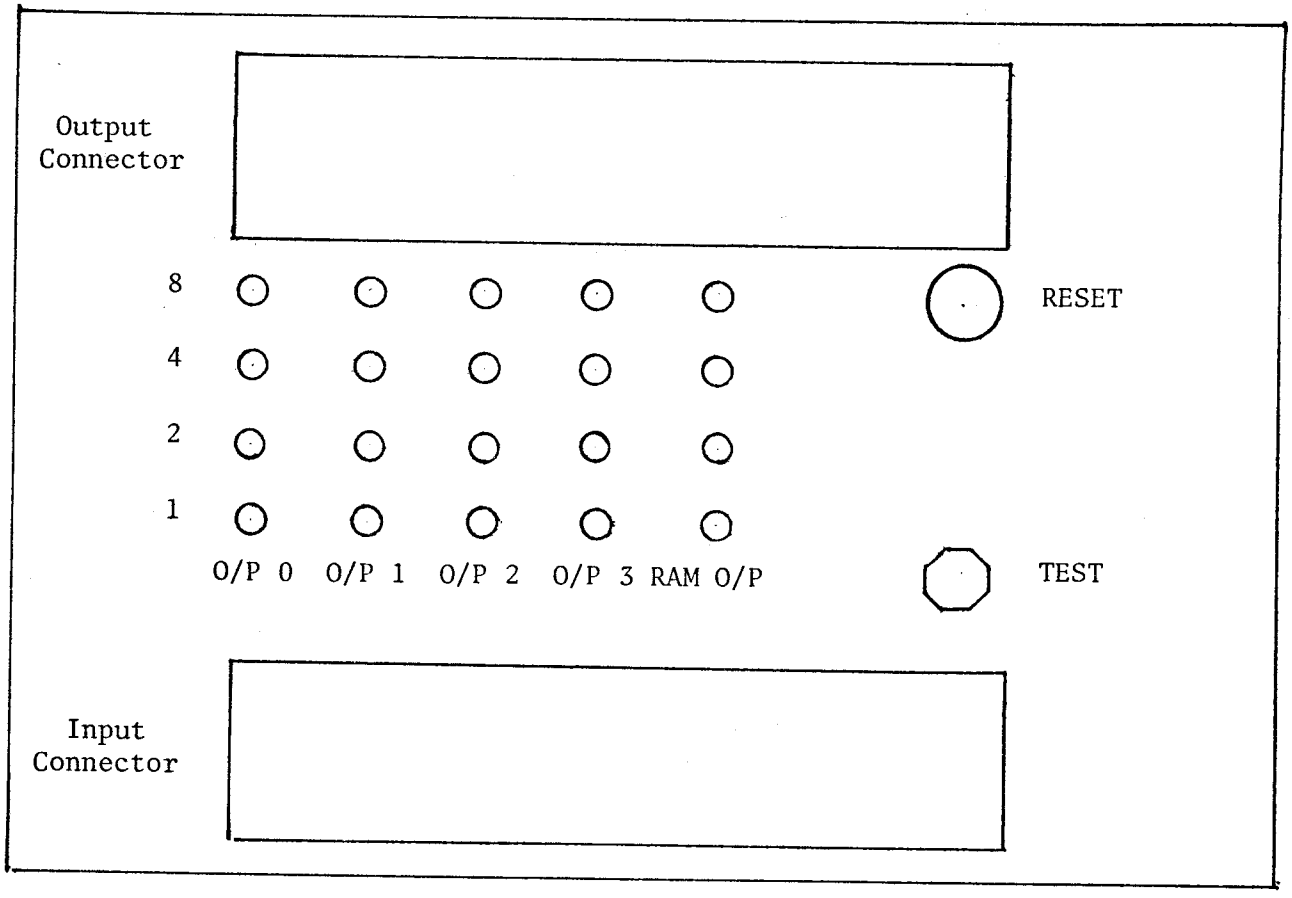


Fig. 3.5 Layout of Display Card

OUTPUT CONNECTOR

		1	2	4	8		1	2	4	8					
GND			Output 0					Output 2							V_{DD}
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
		1	2	4	8		1	2	4	8		1	2	4	8
V_{CC}			Output 1					Output 3				RAM Output			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

V_{CC} = + 5 volts
V_{DD} = -10 volts

INPUT CONNECTOR

		1	2	4	8		1	2	4	8					
GND			Input 0					Input 2							V_{DD}
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
		1	2	4	8		1	2	4	8					
V_{CC}			Input 1					Input 3				TEST			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Fig. 3.6 Scheme of Input/Output Connections

3.5.2 Reset Circuit

A RESET push button is included on the LED matrix card, and is used to switch the levels at the reset input between 0 and +5 volts. The circuit used is shown in Fig. 3.7

3.5.3 TEST Circuit

This circuit produces a single pulse, having levels +5 and -10 volts, and is used in reading out the Fourier coefficients. The circuit, shown in Fig. 3.8, consists of two parts. The cross-connected NAND gates form an R-S flip-flop which eliminates switch bounce, because the output of the flip-flop changes only when the input switches between "1" and "0", and not when the switch bounces. The output of the flip-flop is fed to a level shifter, which changes the levels of the output to +5 and -10 volts. The push button, which is of the momentary contact type, serves to invert the logic level of the output, as set by the switch. The flip-flop eliminates bounce in the push button, too.

The circuit is built on a card which plugs into the input connector. The power supply for the circuit is picked up from the connector, and the output is connected to the "TEST" input through the connector.

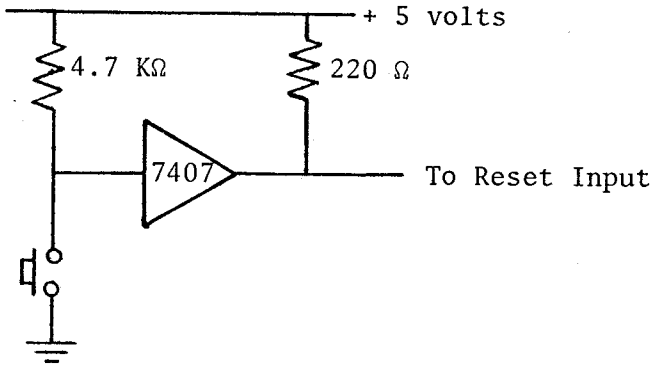


Fig. 3.7 The Reset Circuit

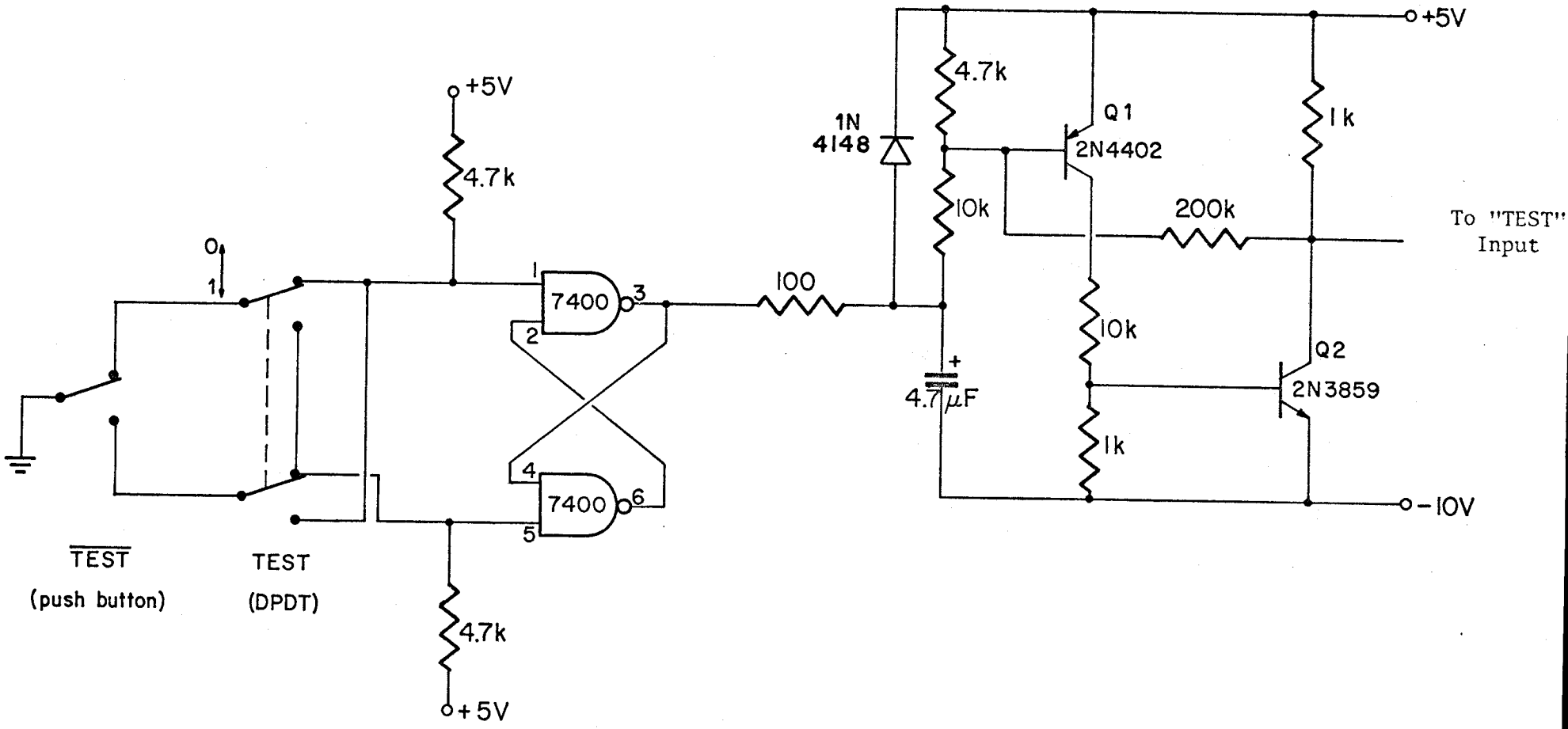


Fig. 3.8 The Test Circuit

CHAPTER IV

THE CONVERSION PROGRAM

4.1 Introduction

The conversion program calculates the first 16 cosine and 16 sine coefficients of the Fourier spectrum of a frequency limited signal from the first 16 cal and 16 sal coefficients of the Walsh spectrum. A brief description of the major steps involved in the conversion is given below, while a detailed explanation of the program is given in later sections. A flow chart for the process is shown in Fig. 4.1.

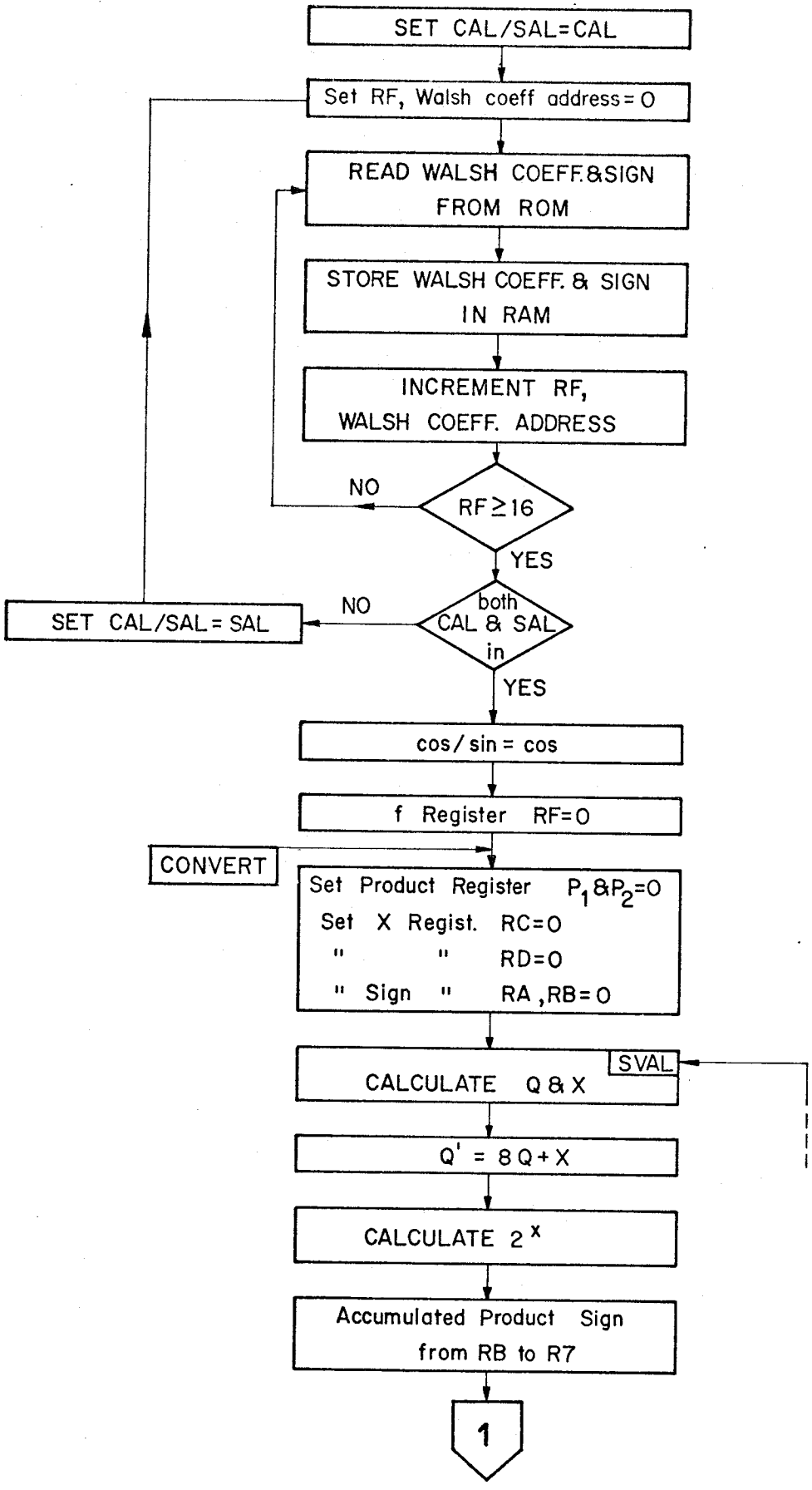
4.1.1 Inputting the Walsh coefficients

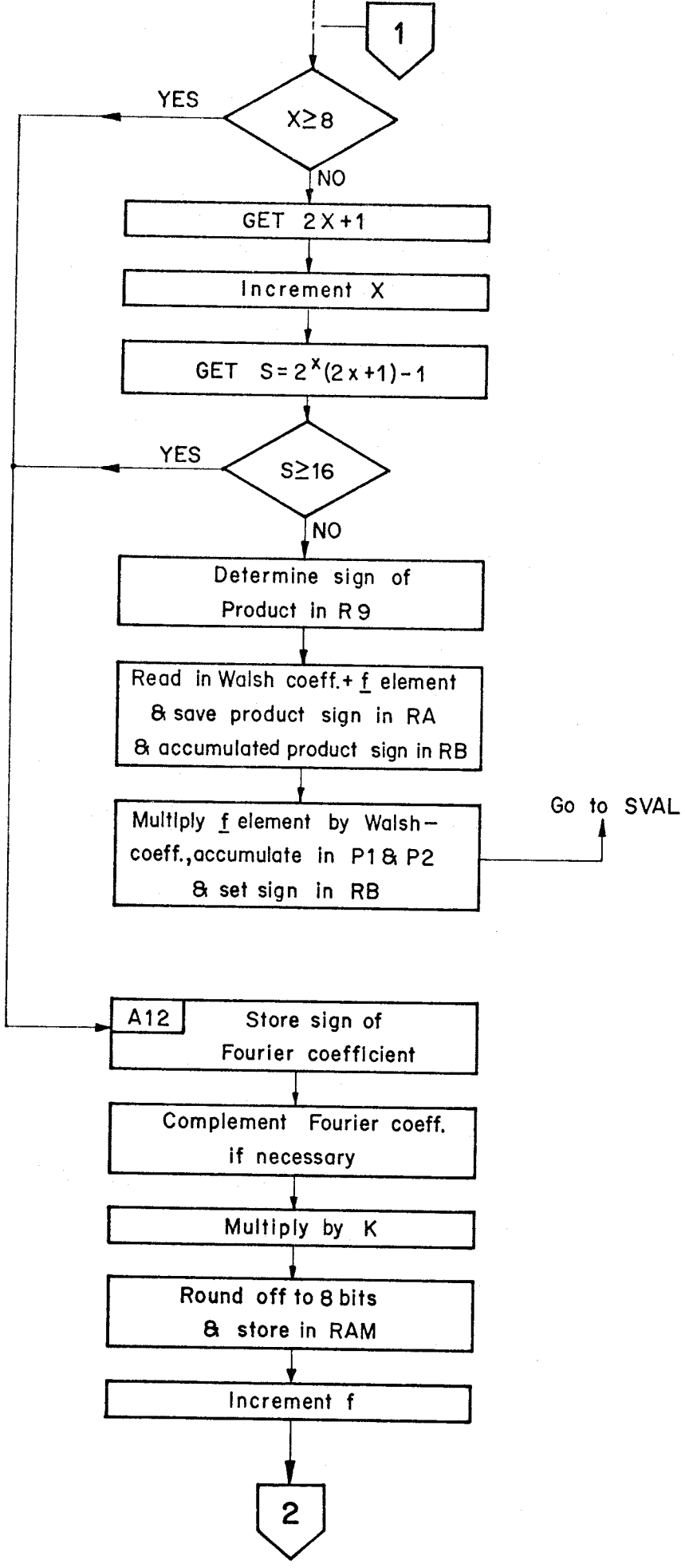
The known Walsh coefficients are presented to the instrument, in Hexadecimal code, by entering them in a read only memory (ROM), using a ROM programmer. These coefficients are then read into the random access memory (RAM), from which they will be fetched during the conversion.

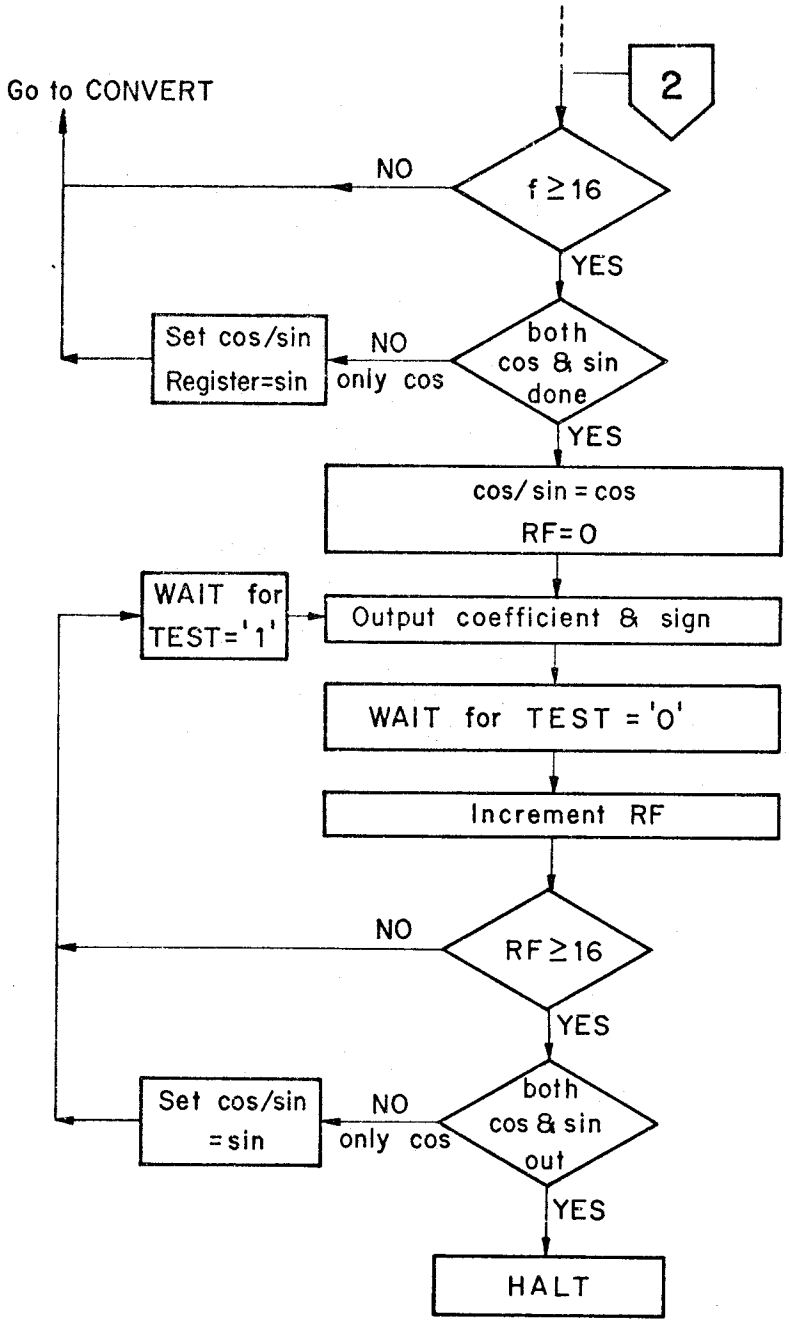
4.1.2 The Walsh-Fourier Conversion

The cosine functions are evaluated first, and then the sines. A counter is set up to keep track of the number of the coefficient being evaluated, and from this the first set of Walsh coefficient and conversion element to be multiplied is selected. The signs of these numbers are read into the index registers, and the resultant sign decides whether the product will be added or subtracted during accumulation. The Walsh coefficient and the conversion element are then read into the index registers, multiplied, and the product accumulated. The next set of Walsh coefficient and

Fig. 4.1 Flow Chart for the Conversion Program




1
YES
X≥8
NO
GET 2X+1
Increment X
GET S=2^X(2x+1)-1
YES
S≥16
NO
Determine sign of Product in R9
Read in Walsh coeff.+ f element & save product sign in RA & accumulated product sign in RB
Multiply f element by Walsh-coeff.,accumulate in P1&P2 & set sign in RB
Go to SVAL
A12
Store sign of Fourier coefficient
Complement Fourier coeff. if necessary
Multiply by K
Round off to 8 bits & store in RAM
Increment f
2
Computation of Fourier coeff.
Compensation for Truncation


2
Go to CONVERT
NO
f ≥ 16
YES
Set cos/sin Register=sin
NO only cos
both cos & sin done
YES
cos/sin = cos
RF=0
WAIT for TEST='1'
Output coefficient & sign
WAIT for TEST='0'
Increment RF
NO
RF ≥ 16
YES
Set cos/sin =sin
NO only cos
both cos & sin out
YES
HALT

conversion element is then selected, and the above procedure repeated. When all the products required to form one Fourier coefficient have been accumulated, the result is multiplied by the K matrix element, and stored in the RAM. The counter is now incremented, and the whole procedure repeated for the other Fourier coefficients.

4.1.3 Outputting the Fourier Coefficients

The Fourier coefficients are read from the RAM and displayed on the LED matrix, by writing on the output ports. Each coefficient is maintained at the port till a pulse is given to the "TEST" input, when the next coefficient is displayed. When all the the coefficients have been displayed, the program halts.

4.2 Storage of Walsh and Fourier Coefficients

The Walsh coefficients are entered into ROM # 3, with locations 0 to 15 storing the cals and locations 16 to 31 the sals. Locations 32 to 47 hold the signs of the cals and 48 to 63 store the signs of the sals. The program then reads these coefficients into the RAM.

Each coefficient is 8 bits long. These coefficients, and the K matrix and conversion elements, are floating point numbers. However, the position of the binary point is always fixed; hence it is ignored, and the number is treated as an integer. The Hexadecimal code which is entered in the ROM (as given in tables 2.1, 2.2, and 2.3) corresponds to the integer formed by ignoring the binary point.

All the Walsh coefficients are scaled so that their magnitude is always less than 1. It will be noticed from table 2.1 that the most significant bit of the Walsh coefficients is always zero, i.e. only 7 bits

are used. The reason for this is given in section 4.3. The position of the binary point in the computed Fourier coefficients is also calculated in section 4.3.

Two RAM chips are required for storing the Walsh and Fourier coefficients. Chip # 0 is used to store the cal and cosine coefficients, and chip # 1 is used for sals and sines. The advantages of this arrangement are:

- if only cals or only sals are involved in the conversion, then only one chip is required;
- the addressing logic for accessing the coefficients is simplified. The counter which keeps track of whether cals or sals are being processed also serves to select the RAM chip.

The method of storing the coefficients in the RAM can be understood with the help of Fig. 4.2. This shows the arrangement for chip # 0. The arrangement for chip # 1 is identical.

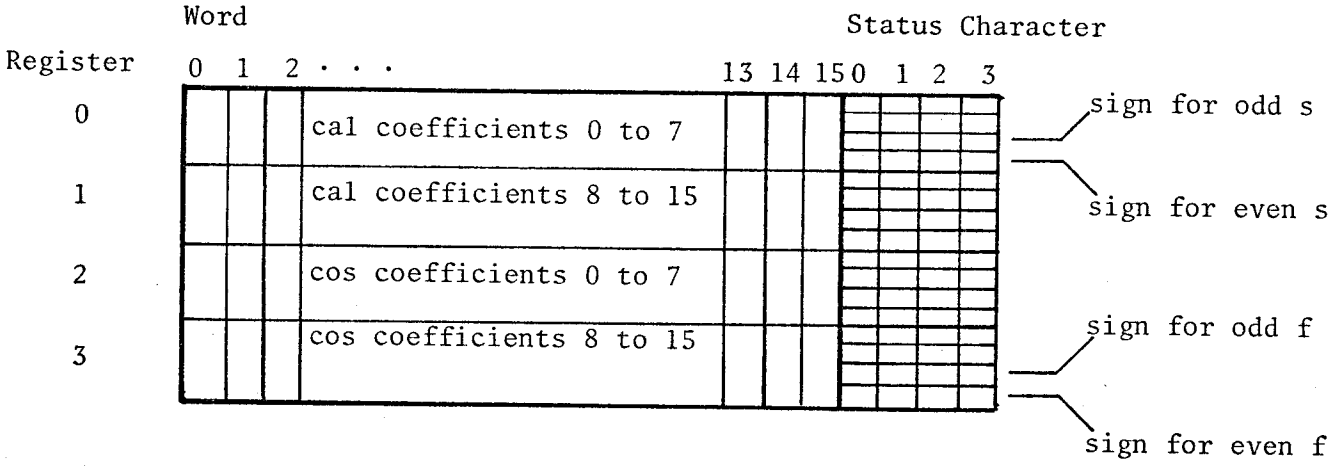


Fig. 4.2 Cal/Cos Storage RAM (Chip # 0)

The 4002 RAM is organized into four registers, numbers 0 to 3. Each register contains 16 4-bit words, and 4 4-bit status characters. Registers 0 and 1 are used to store Walsh coefficients, and registers 2 and 3 are used for Fourier coefficients. Two 4-bit words are required for each coefficient. Thus words 0 and 1 of register 0 store the cal 0[1] coefficient, words 2 and 3 the cal 1 coefficient, and so on.

The signs of the coefficients are stored in the status characters, with each character holding two signs. Thus, the least significant bit (LSB) of register 0 status character zero is used to store the sign of the cal 0 coefficient, and the next higher order bit is used to store the sign of the cal 1 coefficient. If the sign is positive, the bit is kept zero. If the sign is negative, it is set to one.

4.3 Maximum Possible Size of a Computed Fourier Coefficient

The Fourier coefficients are calculated from the relation:

$$a_f = K_{f,f} \cdot \sum_{s=0}^{15} F_{f,s} \cdot A_s$$

To find the maximum value of a_f, we first take the worst case maximum for the summation, i.e. A_s = 1111111, for all s, the signs being such that all terms add up. Therefore, we should sum up the elements in each row of $\underline{F}$, and multiply by 1111111, to get the maximum possible value of the sum for each value of f.

The sum of the elements in each row of the $\underline{F}_{64}$ matrix is shown in

---

1. The arguments 0, 1 etc., correspond to values of s, as defined in section 2.4.1.2, and represent sequency values of 1, 2, etc.

Fig. 4.3. It can be seen that this sum can require 9 bits. The Walsh coefficients are, therefore, chosen to be 7 bits long, so that the product may fit within 16 bits, which is equivalent to four 4-bit registers in the microprocessor.

Row No.	Binary	Decimal	Row No.	Binary	Decimal
0	100101111	303	4	101010111	343
1	110010011	403	5	101101011	363
2	110101011	427	6	100110011	307
3	101101001	361	7	11001110	205

Fig. 4.3 Sum of Elements in Each Row of $\underline{F}_{64}$ Matrix

Each of these sums is now multiplied by the corresponding $\underline{K}$ element, and by A_s = 1111111, taking care that these sums are for alternate values of f. Of course, for the intermediate values, the sums will be less, and need not be considered. The result of the multiplication of the sums by $K_{f,f}$ is shown in Fig. 4.4.

Row No.	$K_{f,f} \cdot \sum F_{f,s}$ (decimal)	Row No.	$K_{f,f} \cdot \sum F_{f,s}$ (decimal)
0	19392	4	28812
1	26598	5	34848
2	29463	6	34998
3	27075	7	2870

Fig. 4.4 $K_{f,f} \cdot \sum F_{f,s}$ for Each Row of $\underline{F}_{64}$ Matrix

The seventh row can, thus, produce the largest possible coefficient, and this corresponds to f = 14. Now, multiplying by $A_s = 1111111_2 = 127_{10}$, we get the largest possible coefficient as 34998 x 127 = 4444746. In binary, this becomes:

$$\boxed{100_{\uparrow}00111}1010010010010 \qquad \text{(23 bits)}$$

This number has been obtained by using integer values for A_s, $F_{f,s}$ and $K_{f,f}$. The effect of ignoring the binary point is one of multiplying A_s by 2^7, $F_{f,s}$ by 2^7 and $K_{f,f}$ by 2^6. Therefore, the computed Fourier coefficient has to be divided by 2^{20}. In effect, the binary point has to be placed 20 positions from the right, as indicated by the arrow. The decimal equivalent of this number is, therefore, 4.2388.

After each Fourier coefficient is computed to 23 bits, the 8 bits shown boxed will be picked out and rounded to the nearest bit. The largest possible value of any coefficient, correct to 8 bits, is

100.01000 (decimal 4.25)

4.4 Storage of $\underline{F}_{64}$ and $\underline{K}$ Matrices

The matrices of constants are stored in ROM # 2. Each ROM word is 8 bits long, and only one word is needed to store each 8-bit constant. The $\underline{F}_{64}$ matrix is stored, one row at a time, in locations 0 to 63. Locations 64 to 79 are used for storing the 16 diagonal elements of the $\underline{K}$ matrix.

Locations 80 to 143 contain the signs for the cal-cos conversion matrix, while locations 144 to 207 contain the signs for the sal-sin conversion matrix. Again, a zero represents a plus, and a one a minus.

4.5 The Conversion Program

In the following description, numbers within parantheses refer to program addresses. The first number refers to the page (each ROM chip is one page of program memory) and the other two numbers refer to the location within that page. The program is listed in Appendix III.

4.5.1 The Input Routine

The program begins at location zero of page zero. The first instruction is NOP (no operation), which avoids any power-on reset problems.

The input routine starts at BEGIN (001). The program jumps to subroutine WALSH[1], which reads the contents of the ROM location on page 3, specified by register pair P0, into register pair P5. Initially, P0 is 0, so that ROM location 0, which contains cal coefficient 0, is read into P5.

Register pair P6 is a counter which selects the RAM location into which the Walsh coefficient is to be written. It is initially at 0, thus selecting RAM chip # 0, register 0, word 0. The SRC instruction (003) selects this location. The high order 4 bits, which are in register RA, are loaded into the accumulator (004) and then written into the selected memory location (005). P6 is then incremented to select the next location, and the above procedure is repeated for the low order 4 bits (006-009).

Register R0 is now incremented twice, thus adding 16 x 2 = 32 to P0. The sign of any Walsh coefficient is stored 32 locations below its magnitude, and thus adding 32 to P0 selects the address of the sign of the Walsh coefficient. The program then jumps to subroutine WALSIGN, which reads the sign word into P3. P0 is then restored to its earlier value, by

---

1. All subroutines are described in section 4.6.

twice decrementing R0 (00E-011). Subroutine SIGNIN then writes the sign into the correct RAM status character.

Register pair P6 is again incremented to select the next RAM location, which will receive the high order 4 bits of the next Walsh coefficient. Similarly, P0 is incremented to set the ROM address of the next Walsh coefficient (014-019).

Register RF is a counter, taking values 0 to 15, which keeps track of the number of the current cal or sal coefficient being transferred. It is incremented and tested for zero. If it is not zero it indicates that the transfer is not complete, and the program returns to BEGIN to read the next Walsh coefficient.

If RF is zero, it indicates that either the first 16, or all 32, transfers have taken place. To determine which, we load the accumulator with register RC and decrement by two (01C-01E). If only the cals have been read in, RC is 0010, and the accumulator becomes zero. Therefore, on testing the accumulator for a non-zero, control transfers to location 021, which sets RC to 4, i.e. 0100 binary. The SRC instruction (003) then selects RAM chip # 1, so that all the sals are transferred to chip # 1. When 16 sal transfers are complete, RC is 0110, and decrementing by two (01C-01E) does not make the accumulator zero. Thus, on testing the accumulator for non-zero (01F), control transfers to CONVERT (025), which proceeds to compute the Fourier coefficients.

4.5.2 The Conversion Routine

At the start (location CONVERT), certain registers, which will hold information calculated by the routine, are cleared (025-02C).

These registers are:

Register RC, which will hold X;

Register RD, which will hold x;

Register RB, which will hold the sign of the accumulated products;

Register RA, which will hold the sign of each product.

Register RE is a flag register which indicated whether cosines or sines are being evaluated. Initially it is 0, for the cosine computation. After all cosines are calculated, it will be set to 2, for the sine computation.

Register RF, which takes values 0 to 15, is a counter representing which cosine or sine coefficient is being evaluated. In other words, it gives the current value of f

For each value of f, up to 8 sets of Walsh coefficients and conversion elements will be selected, multiplied, and then accumulated, to form one Fourier coefficient. This will then be compensated for truncation and stored in the RAM

After one Fourier coefficient has been stored in the RAM, the program will return to CONVERT, to clear registers RA to RD, before calculating the next Fourier coefficient.

4.5.2.1 Selecting the Walsh Coefficient (s value) and Conversion Element (Q value)

Knowing f, the next step is to calculate Q and x, using equation 2-21. Q will be stored in register R1, and x in register RD. Owing to the limited number of index registers, P0 will be used later in the program for multiplication, and therefore the value of Q cannot be saved.

Similarly, RD is used later in the program, and x, too, cannot be saved. Hence, after the multiplication of one Walsh coefficient by its corresponding $\underline{F}$ element, Q and x have to be re-calculated for the next multiplication. This would not be necessary if more index registers were available, because Q and x could be saved, and calculated only for a new value of f. After each multiplication the program loops back to SVAL (02D) and clears P0 and RD to receive Q and x.

To find Q, f is loaded into the accumulator, which is rotated right. For every 1 in the LSB, as detected by a 1 in the carry after the right shift, RD is incremented. When a zero is detected in the carry, rotation stops, the accumulator contains Q, and RD has x. The value of Q is then stored in R1 (031-039).

To illustrate this, consider the example of section 2.4.1.2, i.e. f = 1011. When this is loaded into the accumulator, and rotated right, the accumulator and RD will have the following values:

	Accumulator	Carry (after rotation)	RD
Initial value	1011	-	0
First rotation	0101	1	1
Second rotation	0010	1	2
Third rotation	0001	0	2
		(rotation stops)	

We can see that the accumulator contains 1, which is the value of Q, and RD is 2, which is the value of x.

Since each row of the $\underline{F}_{64}$ matrix occupies 8 locations in the ROM, the value of Q is multiplied by 8, to get the ROM address of the first

element in that row. This is done by jumping to subroutine EIGHT.

Register RC contains X, which represents the position along the $\underline{F}_{64}$ row of the current element to be multiplied. Initially it is zero, and it is incremented after each multiplication. RC is now added to R1 (03C-03E), to give the ROM address of this matrix element. This address is called Q'. The element itself is not read into the registers, as this would block one register pair, which is needed for other calculations. The reading is done only after the sign of the product is evaluated, when the registers used to store the signs of the Walsh coefficient and the matrix element become free.

Having found Q and x, we now have to find the value of s (that is to choose the Walsh coefficient) for the current value of X, from equation 2-22. This is rewritten below as:

$$s = 2^{x}.(2X + 1) - 1 \qquad (4\text{-}1)$$

Therefore, 2^{x} and 2X + 1 are calculated, multiplied, and the product decremented by 1.

To obtain 2^{x}, the one's complement of x is obtained and stored in R7. Register R9 is set to 1, and R7 is incremented and tested for zero. If it is not zero, R8 and R9 are shifted left one place, and R7 is again incremented and tested for zero. When R7 becomes zero, R8 and R9 contain 2^{x}, and the program exits from the loop (03F-047). Another way of doing this would be to take the two's complement of x, and to increment it after the left shift. The disadvantage of this method is that when x = 0 (i.e. for all even values of f) the two's complement of x is 0, and 16 left

shifts are required to obtain the result. With the one's complement method, no left shifts are required, since $2^0 = 1$, which is the starting value of R9. The one's complement method, however, requires more memory space.

The sign of the accumulated products, which is held in RB, is transferred to R7 (050-051), because RB will be used to multiply 2^X and 2X + 1.

Register RC is loaded into the accumulator and shifted left. If the carry is found to be 1, it indicates that X was 8. This means that all multiplications and accumulations necessary to form one Fourier coefficient are over, because the value of X for the last element in an $\underline{F}_{64}$ row is 7. The program then jumps to address A12, to multiply the Fourier coefficient by $K_{f,f}$.

If the carry is not zero, the accumulator is incremented to obtain 2X + 1, which is then stored in R6. RC, i.e. X, is then incremented for the next calculation of s (057-059).

P5 is now cleared (05A) to receive the product of 2^X and 2X + 1, and the multiplication is performed by subroutine MULT. Register RB is loaded into the accumulator, decremented, and then exchanged with RA, which will now contain the value of s. We have to check if this value lies between 0 and 15. If the accumulator is 0, then obviously s must be less than 16, and it can be used to select a Walsh coefficient. If the accumulator is not 0, then s + 1 must be 16 or more. However, if s + 1 is 16 (i.e. s = 15) then RB must have been 0, and decrementing the accumulator when it was loaded with RB would set the carry to zero. If s + 1 was 17 or more, then the carry would be set to 1. Therefore, if the accumulator, when loaded with RA is found to be non-zero, the carry is checked. If it is zero,

we have a valid s, which is used further (05E-064). If the carry is one, it implies that the matrix multiplication for the current value of f is complete, and the program exits from the loop to address A12 (091), to multiply the coefficient by $K_{f,f}$.

If $s \leq 15$, it is multiplied by 2, and stored in register RB (065-068). This is necessary because each Walsh coefficient occupies two addresses in the RAM. The contents of register RE are also shifted left one place (069-06B) and stored in register RA, thus putting the correct RAM chip number (depending on whether cosines or sines are being evaluated) in the most significant bits of RA. In addition, any overflow which occurred on multiplying s by 2 is shifted into the LSB of RA, and this gives the register number within the chip. Thus P5 now has the complete address of the high order 4 bits of the current Walsh coefficient.

For example, let us take s = 13 (1101 binary) during the computation of a sine, so that RE is 0010. Register RB, on shifting left, becomes 1010, and the carry is set to 1. When RE is shifted left, the accumulator becomes 0101, and this is stored in RA. The complete address of the Walsh coefficient is then 0101 1010, which specifies chip # 1, register 1, word 10. This indeed is the location where the 13^{th} sal coefficient is stored.

The SRC instruction to select this RAM location is now executed. However, the coefficient is not read in immediately, for the same reason as in the case of the F_{64} element. Rather, the sign of the product is first evaluated.

4.5.2.2 Determining the Sign of the Product

The signs of the Walsh coefficient and the conversion element are now obtained and combined to form the sign of the product.

The sign of the conversion element is stored 80 or 144 locations below its magnitude, depending on whether the cal-cosine or sal-sine conversion is involved. Therefore RE is checked to determine which conversion is being executed, and then 80 or 144 is added to Q'. This is accomplished by adding 80/16 = 5 or 144/16 = 9 to the high order part of Q' (06D-075). Subroutine SIGN is then used to read the sign into P4, and to restore Q' to its earlier value.

Subroutine SIGNOUT reads the sign of the Walsh coefficient into R8, which is loaded into the accumulator. The carry, which is set to zero if s is even and to 1 if s is odd, by SIGNOUT, is checked for a zero. If it is 1, the accumulator is shifted right to bring the sign into the LSB. If the carry is zero, no shift is performed, because the sign for an even numbered Walsh coefficient is already in the LSB (refer section 4.2).

The accumulator and R9 are added together, and the LSB of the accumulator represents the sign of the product, as can be seen from the truth table below:

Walsh Coeff. Sign	Accumulator Contents	Conversion Element Sign	R9 Contents	Product Sign	Accumulator Contents
+	0000	+	0000	+	0000
+	0000	-	0001	-	0001
-	0001	+	0000	-	0001
-	0001	-	0001	+	0010
					↑ LSB

The product sign is stored in R9, and will be used during the multiplication-accumulation to decide whether to add or subtract.

4.5.2.3 Formation and Accumulation of Products

The high order 4 bits are read from the RAM location selected by the SRC instruction at 06C, and are stored in R6. RB is then incremented to select the next RAM location, and an SRC instruction is executed. R7, which holds the sign of the accumulated products, is saved in RB, which no longer needs to be preserved. The low order 4 bits of the Walsh coefficient are then read into R7 (081-088).

R9, which holds the product sign, is saved in RA, which too need not be saved any longer. Subroutine READF then read the conversion element into R8 and R9 (089-08C).

Subroutine MULTIPLY is now executed. Registers R8 and R9 are multiplied by registers R6 and R7, and the product is either added into or subtracted from register pairs P1 and P2, depending on the sign of the product. The sign of the accumulated products is set in RB. P0 is destroyed during the multiplication.

The program now loops back to SVAL to calculate new values of Q, x and s. If $s \geq 16$, or $X \geq 8$, then the program exits from the loop to address A12, to multiply the Fourier coefficient by the compensation element. The sign of the coefficient is in R7, at this stage.

4.5.2.4 Compensation for Truncation and Storage of the Coefficient.

The sign of the Fourier coefficient is stored first. Subroutine SELECT selects the RAM chip and register in which the sign is to be stored, and also the word in which the high order 4 bits of the Fourier

coefficient will be stored. The address of the word is stored in P0.

Register R1 is transferred to RD, which serves as an argument for subroutine SIGNIN. This subroutine takes the sign from R7 and puts it into the correct status character of the RAM.

Register R7 is then checked for a zero (097-099). If it is 1, then the Fourier coefficient is negative (in two's complement form), and has to be recomplemented to obtain it in the true form (09A-0AC).

The $\underline{K}$ matrix elements are stored from locations 64 to 79 in ROM # 2. Therefore, the address of the compensation element is generated in P0 by adding 64 to the f value. Subroutine FETCHK is then used to read this element into P3 (0AD-0B2).

Register pairs P4, P5, P6 are cleared to receive the final Fourier coefficient, and P0 is also cleared to receive the bits shifted out of P1 and P2, when they are shifted left during the multiplication. The multiplication is performed by subroutine MUL (0B3-0BC).

Of the 24 bits the product occupies, we have to pick out the 3 least significant bits of RC, which represent the integer portion, and all of RD and the MSB of RA. Therefore, RA is loaded into the accumulator and rotated left, so that the MSB comes into the carry. The accumulator is then exchanged with RD, and again rotated left. The carry comes into the LSB and the MSB goes into the carry. The accumulator now contains the low order 4 bits of the result. It is exchanged with RC and again rotated left. The 3 least significant bits of RC move up one place, and the carry moves into the LSB. Now, the accumulator contains the high order 4 bits of the result. It is exchanged with RD, which contains the remaining 3 bits of

the original RA. The MSB is shifted left into the carry, and tested for zero. If it is zero, the result does not have to be rounded up to the next higher number. If it is one, the result is incremented by 1. The high order 4 bits are now in RD and the low order 4 bits in RC (0BD-0C9).

To store the results in the RAM, RD is loaded into the accumulator and written into the memory location already selected (0CA-0CB). The address of this location is again generated in P0 by subroutine SELECT. R1 is then incremented and an SRC instruction executed to select the next the next RAM location. RC is then written into this location (0CC-0D1).

Finally, subroutine CHECK checks if all the Fourier coefficients have been calculated and stored. If not, it increments RF to the next value of f, ands sets the accumulator to zero. When all the cosines have been evaluated, it resets RF to zero and sets RE to 2 and the accumulator to zero. If all computation is over, it sets the accumulator to 1. The accumulator is then tested, and if found to be zero, the program returns to CONVERT, to repeat the whole process for finding the next Fourier coefficient. If the accumulator is 1, the program proceeds to output the Fourier spectrum.

4.5.3 Output Routine

Register pair P7 is intiallized to zero, and serves as a counter for the Fourier coefficients, in the same way as in the conversion routine.

Register pairs P1. P2, P3, and P4 are set to (1,0), (2,0), (0,0) and (3,0) to select output ports 1, 2, 0 and 3 respectively. The accumulator, which contains 1, is written on port 3. This serves as a flag to indicate that the computation is complete (0D6-0E1).

Subroutine SELECT then selects the RAM address of the high order 4 bits of the Fourier coefficient, which is read in and written on port 1 (0E2-0E6). The low order 4-bit address is then selected (0E7-0E8). However, these 4 bits are not read in or outputted at this point, because, in selecting the output port, the RAM chip selection would be lost. Then, to read the sign from the RAM would again require RAM selection. Therefore, the outputting of these bits is done after the sign is read in. RF is shifted left and placed in RB. This serves as an argument for subroutine SIGNOUT, which reads the sign of the Fourier coefficient into R8 (0E9-0ED). Again, the sign is not written on an output port at this stage, as this would destroy the RAM selection.

The low order 4 bits are then read in and written on port 2 (0EE-0F0).

The sign of the coefficient is then transferred to the accumulator. Subroutine SIGNOUT sets the carry to zero for an even numbered coefficient and to 1 for an odd numbered coefficient. The carry is therefore checked, and if it is one, the accumulator is shifted right to bring the sign into the LSB. If the carry is zero, the sign is already in the LSB, but the next most significant bit contains the sign of the next higher order (odd numbered) coefficient, and we do not want this to appear on the output port. Therefore, the accumulator is shifted right, so that the sign goes into the carry, the accumulator is cleared by loading it with R3 (which contains zero), and then the sign is shifted left back into the accumulator LSB (0F1-0FA). Output port 0 is then selected, and the sign written on it.

The "TEST" input is checked for a zero, which is the sign for the

program to proceed. If "TEST" is "1", the program goes into a waiting loop till "TEST" becomes "0". The reason for this is explained below.

If "TEST" is "0", the program executes subroutine CHECK, which, as in the conversion routine, checks if all coefficients have been outputted. If so, the accumulator is set to 1, and the program enters an infinite loop, i.e. in effect it halts. If the accumulator is zero, the program must return to address STAR to output the next coefficient. However, if there were no control over this looping, the program would output the coefficients so fast that the operator would not be able to see any of them. Therefore, the program is not allowed to return to STAR unless the operator sets "TEST" to "1" by pushing the TEST button. Now, if the first check of the "TEST" input were not present, and "TEST" remained at "1", again all coefficients would be outputted at a very high speed. By inserting two checks for "TEST", we ensure that the program will step from one coefficient to another only when the "TEST" input is set to "1" and then again reset to "0".

4.6 The Subroutines

4.6.1 WALSH, WALSIGN, READF, FETCHK, SIGN

These subroutines (located at 340, 342, 2DB, 2DF, and 2D0, respectively) are used to read data from a ROM into a specified index register. They all use the FIN (fetch indirect) instruction, and have to be on the same page of memory as the data to be read. Subroutine SIGN also restores the value of Q', by subtracting 80 or 144, to that necessary for reading the corresponding $\underline{F}_{64}$ matrix element. Subroutine READF clears register pair P0, after the read operation, to prepare it for the multiplication

subroutine.

4.6.2 EIGHT

This subroutine (location 10F) multiplies the value of Q by eight, and places the result in P0. The multiplication is effected by shifting Q left three times.

4.6.3 MULT, MULTIPLY, MUL

These three subroutines (locations 11C, 1CA, and 146, respectively) perform multiplication. If more index registers were available, it would be possible to reserve certain registers in which arguments for multiplication could be passed, and only one subroutine would be required. However, as this is not possible, three subroutines are necessary to handle the different registers in which the arguments occur.

On entering the multiplication subroutine, the program jumps to another subroutine, such as 6→ (location 344) or 67→ (location 12E), which shifts the LSB of the multiplier into the carry. If the carry is 1, the multiplicand is added to the product registers; if it is zero, the addition is skipped. The right shifted multiplier is tested for zero. If it is, the multiplication is complete; if it is not, the multiplicand is shifted left one place by another subroutine, such as 89← (location 107) or 0189← (location 136) or 014523← (location 349), and the procedure of testing the LSB of the multiplier is repeated.

One feature of this technique of multiplication is the testing of the multiplier, after each right shift, for zero. This procedure can save a lot of time, compared to one where a counter is set up to shift the multiplier right as many times as there are bits in the multiplier. It is

particularly useful when one of the numbers to be multiplied can be expected to be zero, and can be made the multiplier, rather than the multiplicand. For example, a number of the Walsh coefficients will be zero, while none of the $\underline{F}_{64}$ elements is zero. Therefore, the matrix element is made the multiplicand, while the Walsh coefficient is made the multiplier. An exception to this arrangement is the multiplication of the uncompensated Fourier coefficient by the $\underline{K}$ factor. Many of the Fourier coefficients can be expected to be zero, and by the above logic one should make the Fourier coefficient the multiplier. However, this is a 16-bit number, and in the worst case it could involve 16 right shifts and additions. The $\underline{K}$ element is 8 bits long, and if it is made the multiplier a maximum of 8 right shifts and additions need to be performed. Since addition is a relatively slow process, it is preferable to make the $\underline{K}$ element the multiplier. Further, using the Fourier coefficient as the multiplier would require another subroutine for shifting it right, whereas the $\underline{K}$ element can use the same subroutine used for shifting the 8-bit Walsh coefficient.

Subroutine MULTIPLY is slightly different from the other two in that it checks register RA for the sign of the product, and then either adds (using subroutine ADD2, location 1DF) or subtracts (using subroutine SUB2, location 2E1) from the product registers. In ADD2, if, after the addition is complete, an overflow occurs, then the number must have been negative and has changed to positive owing to the addition. The sign register RB is therefore set to zero. Similarly, in SUB2 if no overflow occurs, the number must have been positive, and has changed to negative owing to the subtraction. RB is therefore set to 1. All negative numbers are in two's complement form.

An alternative method for performing the multiplication would be to shift the product registers right after each addition, rather than shifting the multiplicand left. This would eliminate the need of registers to take the bits shifted out of the multiplicand. However, this method would require a counter to keep track of the number of right shifts; secondly, it cannot be used when the products have to be accumulated.

4.6.4 SELECT

This subroutine (location 153) uses the value of f in register RF, and the cosine/sine flag register RE, to select a RAM location for reading from or writing in. The address of the RAM word to hold the cosine or sine is generated in P0 by shifting P7 left one place, incrementing it by 32, and storing in P0. For example, if the address for the seventh sine is required, then P7 is 0010 0111. On shifting left it becomes 0100 1110, and on adding 32 (i.e. adding 2 to the higher 4 bits) the final result in P0 is 01 10 1110. This selects chip # 1 (first two bits), register 2 (next two bits), word 14 (last four bits), which is where the seventh sine coefficient is stored.

4.6.5 SIGNIN

This subroutine (location 182) writes a sign, available in R7, into one of the status characters in the RAM register which holds the corresponding coefficient. The actual status character and bit are selected by the subroutine from the coefficient number. The logic is best illustrated by an example.

Suppose we want to write the sign for a coefficient numbered 0101. The calling program will already have selected the RAM chip and register, and will have placed the coefficient number, shifted left by one place,

in RD, i.e. RD is 1010[1]. Similarly, if the coefficient number were 0100, RD would be 1000. RD is now shifted right twice in the accumulator, so that it is left with 0010 for both the above cases. This is the status character number which holds the sign for both coefficient number 0100 and 0101.

The accumulator is now tested for zero. If it is zero, the selected status character is number zero; if not the accumulator is again decremented and tested for zero. If it is now zero, status character 1 is selected; if not, the accumulator is again decremented, and so on.

Knowing which status character has to be written into, we now have to decide whether to write into the LSB or the next most significant bit. To do this, RF, which is a counter for the coefficient number, is rotated right in the accumulator to check for an even or odd numbered coefficient. If the carry is zero, the coefficient is even numbered, and the sign is directly written into the status character. If the carry is 1, the coefficient is odd numbered, and the sign cannot be directly stored, as it would erase the sign of the even numbered coefficient previously stored in the same status character. Therefore, we first read the status character into the accumulator, rotate it right to save the previous sign in the carry, load the new sign word into the accumulator, and rotate it back left. Now the accumulator can be written into the status character, with the sign of the even numbered coefficient in the LSB and that of the odd numbered

---

1. When the sign of the Walsh coefficient is being stored (location 012), RD is incremented before the sign storage, so that it is, for example, 1011 and not 1010. However, by clearing the carry between right shifts, this 1 can be eliminated.

coefficient in the next most significant bit.

4.6.6 SIGNOUT

This subroutine (location 167) reads the sign of a coefficient from a status character. The number of the coefficient must be in RB. The method of selecting the status character and reading the sign is identical to that in subroutine SIGNIN. In addition, before returning to the main program, RB is rotated right twice, so that the carry is set to 0 or 1 depending on whether the coefficient is even or odd numbered. This will be used by the main program to decide which bit of the status character has the sign.

4.6.7 CHECK

This subroutine (location 15E) is used to increment the Fourier coefficient counter RF. If the counter goes to zero, it indicates that 16 coefficients have been evaluated, and we have to decide if these were the cosines or sines. RE is checked, and if it is zero the sines are yet to be processed. RE is set to 2, and the subroutine returns with the accumulator set to zero. If RF does not become zero on incrementing, the program returns with the accumulator set to zero. If RE is not zero, then the sines have been processed, and the subroutine returns with the accumulator set to 1.

CHAPTER V

DISCUSSION

5.1 Test Results

The signal which was chosen to test the program was:

$$f(t) = \cos t + 0.25\sin t - 1.25\sin 2t + \sin 3t$$

The form of this function was chosen for two reasons:

- Both sines and cosines are present, and thus both sections of the conversion program, and the logic which signals the completion of each conversion, are tested;

- Both positive and negative signs are involved, and therefore the logic for keeping track of the signs is tested.

The Walsh spectrum of this signal has already been given in section 2.3.3. This spectrum was entered into ROM # 3, using a ROM programmer. The entire conversion was executed in about 900 msec. Owing to the drift in the microprocessor's clock frequency, a more accurate timing was not possible. The Fourier spectrum obtained at the output ports is listed in table 5.1.

5.2 Accuracy

The accuracy of the Fourier coefficients depends on the number of bits used to represent the conversion elements and the coefficients. Thus, the only errors to be considered are the truncation errors, which occur as follows:

TABLE 5.1 Results of the Test Program

Cosine Coefficients

Coefficient No.	Magnitude	
	Binary	Decimal
0	001.00000	1.00
1	000.00000	0.00
2	000.00000	0.00
3	000.00000	0.00
4	-000.00000	-0.00
5	000.00000	0.00
6	-000.00000	-0.00
7	000.00000	0.00
8	-000.00000	-0.00
9	000.00000	0.00
10	000.00000	0.00
11	000.00000	0.00
12	000.00000	0.00
13	000.00000	0.00
14	-000.00000	-0.00
15	000.00000	0.00

Sine Coefficients

Coefficient No.	Magnitude	
	Binary	Decimal
0	000.01000	0.25
1	-001.01000	-1.25
2	001.00000	1.00
3	000.00000	0.00
4	000.00000	0.00
5	000.00000	0.00
6	000.00000	0.00
7	000.00000	0.00
8	-000.00000	-0.00
9	000.00000	0.00
10	000.00000	0.00
11	000.00001	0.03
12	-000.00000	0.00
13	000.00001	0.03
14	000.00000	0.00
15	000.00000	0.00

a). Truncation of Walsh Coefficients

Seven bits are used to represent the Walsh coefficients, and the maximum possible error is $\pm \frac{1}{2}$ of 2^{-7}, i.e. $\pm 2^{-8}$.

b). Truncation of Conversion Elements

Again, seven bits are used to represent the fractional part, and the maximum possible error is $\pm 2^{-8}$.

c). Truncation of the Compensation Element

Six bits are used to represent the fractional part, and the maximum possible error is $\pm 2^{-7}$.

At each multiplication of a Walsh coefficient by a conversion element the errors are added. Thus each product can have an error of $2.\pm 2^{-8} = \pm 2^{-7}$. Up to 8 products are needed to form one Fourier coefficient, so the error in the uncompensated coefficient can be $8.\pm 2^{-7}$

On multiplying the Fourier coefficient by the compensation element, the error can increase to $\pm(2^{-7} + 8.2^{-7}) = \pm 9.2^{-7}$

Finally, the Fourier coefficient is truncated to 5 binary bits, and this introduces a further error of 2^{-6}. The total absolute maximum error is, therefore, $\pm(9.2^{-7} + 2^{-6}) = \pm 11.2^{-7} = \pm 0.0859$. Expressed as a percentage of the maximum possible coefficient, this becomes:

$$\pm \frac{0.0859}{4.2388} \times 100 = \pm 2.03\%$$

This absolute error can, of course, form a larger percentage in a coefficient of lower magnitude. On the other hand, there may be no error at all in some cases. For example, the test signal coefficients were obtained without any error, because the various errors cancel each other.

5.3 Execution Time

To compute the execution time the number of instructions have to be counted, since the time for each instruction, 10.8 μsec., is fixed. Of course, care has to be taken that the number of times a loop is executed is accounted for.

First, the worst case number of instructions for execution of each subroutine were counted. These are listed in table 5.2.

Subroutine	No. of instructions	Subroutine	No. of instructions
WALSH	2	ADD	20
SIGN	11	ADD1	8
READF	4	ADD2	18
FETCHK	2	SELECT	11
SUB2	21	CHECK	8
WALSIGN	2	SIGNOUT	27
6→	5	SIGNIN	25
67→	8	EIGHT	27
89←	8	MULT	119
0189←	14	MUL	475
014523←	20	MULTIPLY	489

TABLE 5.2 Number of Instructions in each Subroutine

Next, the number of instructions in each section of the program are computed. The number of instructions in each subroutine are accounted for here. Table 5.3 lists the number of instructions in each section of the program.

TABLE 5.3 Number of Instructions in each Section of Conversion Program

Section	Number of Instructions	Number of times executed	Total
NOP	1	1	1
Inputting Walsh coefficients	54	32	1728
Checking for completion	17	1	17
Initialization for conversion	8	32	256
Computation of each product and accumulation	833	172	143276
Checking for formation of eight products	126	16	2016
Checking for $s \geq 16$	259	16	4144
Compensation for trucation and storage in RAM	601	32	19232
Outputting first coefficient	88	1	88
		Total =	170,758.

The execution time for the program is therefore 170,758 x 10.8 μsec = 1.81 sec. This represents the maximum possible time for the first Fourier coefficient to be displayed at the output port.

5.4 Cost

The approximate cost of the system can be split in the following way:

Programmer	$ 2,000
Debugger	$ 300
PLS-401 card	$ 170
LEDs & accessories	$ 50

In this cost analysis, however, it must be realized that the cost of the

programmer and debugger should certainly not be allotted to this one system alone, since they are used for a variety of purposes. Even the cost of the card need be allotted solely to this system only if it is to be used purely for Walsh-Fourier conversion. Otherwise, by merely replacing the ROMs the system can be used in many different applications, and the great flexibility of a microprocessor system adds to its economy in use.

5.5 Power Requirements

The power requirements of the system are estimated as follows:

PLS-401 card	+5 volts, 550 mA; -10 volts, 350 mA.
7407 buffers (4 nos., 41 mA. each)	+5 volts, 164 mA.
Total	+5 volts, 714 mA; -10 volts, 350 mA.

5.6 Size

The system which was constructed has dimensions of approximately 325 mm. wide x 145 mm. deep x 120 mm. high. The layout, however, was such as to permit easy access to all parts. All the components can be packaged into a much smaller volume.

It is instructive to compare the cost, power requirements and size of this converter with those for the design using standard IC gates proposed by Doran [2]. The figures for that design are:

Cost	$ 1,359
Power requirement	+ 5 volts, 7.031 A. - 9 volts, 50 mA. -12 volts, 52 mA.
Size	305 mm. wide x 127 mm. high x 254 mm. deep

5.7 Extensions and Improvements

5.7.1 Accuracy, Speed and Number of Coefficients

To process a larger number of coefficients, the program would have to manage a counter of more than 4 bits. The logical next higher size is 8 bits, and an 8-bit microprocessor could be used to process up to 2^8 = 256 coefficients. Using an 8-bit microprocessor would also allow using a larger number of bits for each coefficient, thus improving the accuracy. The number of fetches and instruction steps could also be reduced, and this, coupled with the faster instruction cycle time of some 8-bit microprocessors, could improve the speed of execution.

5.7.2 Input and Output

To demonstrate the working of the program, known Walsh spectra were programmed into a ROM. In a practical case this would not be a very suitable way of inputting coefficients. However, the program can be very easily modified to read the coefficients from the input ports. The coefficients could then be presented at the input ports through switches or a keyboard, or the outputs of a Walsh spectral analyzer could be directly interfaced at these ports.

For outputting the coefficients, the output ports could be interfaced to a seven segment display, so as to have a decimal output. The conversion from binary may be accomplished either by using external hardware decoders, or by modifying the program itself. While the program has been written to allow sequential read-out of the coefficients, it can be very simply altered to allow read-out only of selected coefficients. The coefficient to be read out can be set on a rotary switch connected to an input port through an encoder. The program would interrogate the port and then

display the coefficient specified.

5.7.3 Conversion from Fourier to Walsh Spectra

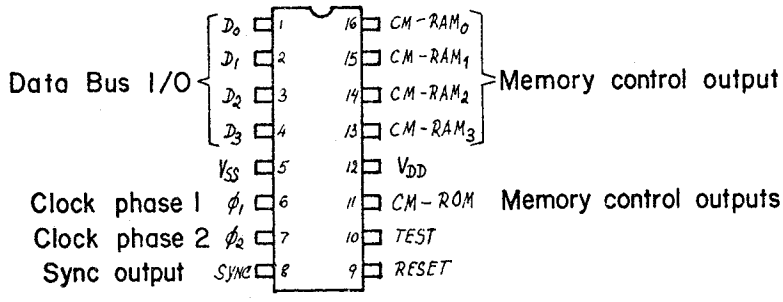
The dual function of converting from the Fourier spectrum of a sequency limited signal to its Walsh spectrum can be readily accomplished. The entire procedure is the same as for the Walsh-Fourier conversion, and only the matrices of constants need to be changed. In effect, only ROM # 2 need be reprogrammed with the matrix values for Fourier-Walsh conversion.

APPENDICES

APPENDIX I

PIN FUNCTIONS OF THE 4004 CHIP

The 4004 is packaged in a 16 pin DIP. The pin configuration is shown in the following figure. A brief functional description of each pin is given below:



Pin No.	Designation	Description of Function
1-4	D_0 - D_3	Bidirectional data bus. All address and data communication between the processor and the RAM and ROM chips is handled by way of these four lines.
5	V_{ss}	Most positive supply voltage.
6-7	ϕ_1 - ϕ_2	Non-overlapping clock signals which determine processor timing.
8	SYNC	SYNC output. Synchronization signal sent by processor to indicate beginning of each cycle.
9	RESET	RESET input. A "1" level applied to this

Pin No.	Designation	Description of Function
		pin clears all flag and status flip-flops and forces the program counter to zero.
10	TEST	TEST input. The logical state of this input can be examined by the JCN instruction.
11	CM-ROM	This pin enables a bank of upto 4K ROM.
12	V_{dd}	Main supply voltage to the processor. Value must be V_{ss} - 15.0 volts ±5%.
13-16	$CM\text{-}RAM_0$ to $CM\text{-}RAM_3$	CM-RAM outputs. These outputs act as bank select signals for the 4002 RAM chips in the system.

APPENDIX II

INSTRUCTION SET OF THE 4004

HEX CODING		MNEMONIC OPR	MNEMONIC OPA	DESCRIPTION OF OPERATION
0	0	NOP		No operation.
1 A_2	C_X A_1	JCN	C_X LABEL	Jump on condition C_X to the program memory address A_1, A_2, otherwise continue in sequence. (see back cover).
2 D_2	P_X0 D_1	FIM D_2	P_X D_1	Fetch immediate from program memory data D_1, D_2 to index register pair P_X
2	P_X1	SRC	P_X	Send register control. Send the contents of index register pair P_X to I/O ports and RAM register as chip select and RAM character address.
3	P_X0	FIN	P_X	Fetch indirect. Send contents of register pair 0 out as a program memory address. Data fetched is placed into register pair P_X .
3	P_X1	JIN	P_X	Jump indirect. Jump to the program memory address designated by contents of register pair P_X
4 A_2	A_3 A_1	JUN	 LABEL	Jump unconditional to program memory address A_1, A_2, A_3.
5 A_2	A_3 A_1	JMS	 LABEL	Jump to subroutine located at program memory address A_1, A_2, A_3. Save previous address (push down in stack).
6	R_X	INC	R_X	Increment contents of register R_X.
7 A_2	R_X A_1	ISZ	R_X LABEL	Increment and step on zero. Increment contents of register R_X, if result is not 0 go to program memory address A_1, A_2, otherwise step to the next instruction in sequence.
8	R_X	ADD	R_X	Add contents of register R_X to accumulator.
9	R_X	SUB	R_X	Subtract contents of register R_X to accumulator with borrow.
A	R_X	LD	R_X	Load contents of register R_X to accumulator.
B	R_X	XCH	R_X	Exchange contents of index register R_X and accumulator.
C	D_X	BBL	D_X	Branch back one level in stack to the program memory address stored by a prior JMS instruction. Load data D_X to accumulator.
D	D_X	LDM	D_X	Load data D_X to accumulator.
E	X	I/O and RAM register instructions		
F	X	Accumulator instructions		

A_1 Low order address bits
A_2 High order address bits
A_3 Chip select

P_X1 Register pairs P_0 through P_7 designated by odd characters 1, 3, 5, 7, 9, B, D, F
P_X0 Register pairs P_0 through P_7 designated by even characters 0, 2, 4, 6, 8, A, C, E
R_X Register 0 ⟶ F
D_X Data
D_1 Data for odd register
D_2 Data for even register
C_X Jump conditions

I/O AND RAM REGISTER INSTRUCTIONS

HEX CODING	MNEMONIC OPR	MNEMONIC OPA	DESCRIPTION OF OPERATION
E 0	WRM		Write the contents of the accumulator into the previously selected RAM register character.
E 1	WMP		Write the contents of the accumulator into the previously selected RAM output port. (Output lines.)
E 2	WRR		Write the contents of the accumulator into the previously selected output port. (I/O lines.)
E 3	WPM		Write the contents of the accumulator into the previously selected RAM program memory.
E 4	WRO		Write the contents of the accumulator into the previously selected RAM status character 0.
E 5	WR1		Write the contents of the accumulator into the previously selected RAM status character 1.
E 6	WR2		Write the contents of the accumulator into the previously selected RAM status character 2.
E 7	WR3		Write the contents of the accumulator into the previously selected RAM status character 3.
E 8	SBM		Subtract the previously selected RAM register character from accumulator with borrow.
E 9	RDM		Read the previously selected RAM register character into the accumulator.
E A	RDR		Read the contents of the previously selected input port into the accumulator. (I/O lines.)
E B	ADM		Add the previously selected RAM register character to accumulator with carry.
F C	RDO		Read the previously selected RAM status character 0 into accumulator.
E D	RD1		Read the previously selected RAM status character 1 into accumulator.
E E	RD2		Read the previously selected RAM status character 2 into accumulator.
E F	RD3		Read the previously selected RAM status character 3 into accumulator.

ACCUMULATOR INSTRUCTIONS

HEX CODING	MNEMONIC OPR	MNEMONIC OPA	DESCRIPTION OF OPERATION
F 0	CLB		Clear both. (Accumulator and carry.)
F 1	CLC		Clear carry.
F 2	IAC		Increment accumulator.
F 3	CMC		Complement carry.
F 4	CMA		Complement accumulator.
F 5	RAL		Rotate left. (Accumulator and carry.)
F 6	RAR		Rotate right. (Accumulator and carry.)
F 7	TCC		Transmit carry to accumulator and clear carry.
F 8	DAC		Decrement accumulator.
F 9	TCS		Transfer carry subtract and clear carry.
F A	STC		Set carry.
F B	DAA		Decimal adjust accumulator.
F C	KBP		Keyboard process. Converts the contents of the accumulator from a one out of four code to a binary code.
F D	DCL		Designate command line.
F E			
F F			

APPENDIX III

THE CONVERSION PROGRAM

PROGRAM ASSEMBLY FORM

TITLE

DATE

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND	COMMENTS
0	00	00		NOP		
	1	53	BEGIN	JMS		
	2	40			WALSH	
	3	2D		SRC	6	
	4	AA		LD	A	
	5	E0		WRM		
	6	6D		INC	D	
	7	2D		SRC	6	
	8	AB		LD	B	
	9	E0		WRM		
	A	60		INC	0	
	B	60		INC	0	
	C	53		JMS		
	D	42			WALSIGN	
	E	A0		LD	0	
	F	F8		DAC		
	10	F8		DAC		
	1	B0		XCH	0	
	2	51		JMS		
	3	82			SIGNIN	
	4	7D		ISZ	D	
	5	17			A1	
	6	6C		INC	C	
	7	71	A1	ISZ	1	
	8	1A			A2	
	9	60		INC	0	
	A	7F	A2	ISZ	F	
	B	01			BEGIN	
	C	AC		LD	C	
	D	F8		DAC		
	E	F8		DAC		
	F	1C		JCN	A1	

WRITING WALSH COEFFICIENTS IN RAM.

INDEX REGISTERS

E	. . .	P7	COEFF. CONT.
C	RAM LOC.	P6	ADDRESS
A	WALSH	P5	COEFF.
8	. . .	P4	. . .
6	WALSH	P3	SIGN
4	. . .	P2	. . .
2	. . .	P1	. . .
0	WALSH COEF.	P0	ADDRESS

WRITING SIGN IN RAM

INCREMENTING COUNTERS AND CHECKING FOR COMPLETION.

PROGRAM ASSEMBLY FORM

TITLE

DATE

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND	COMMENTS
0	20	25			CONVERT	
	1	2C		FIM	6	
	2	40		4	0	
	3	40		JUN		
	4	01			BEGIN	
	5	2C	CONVERT	FIM	6	
	6	00		0	0	INITIALLIZE
	7	24		FIM	2	STORAGE REGISTERS
	8	00		0	0	
	9	21		FIM	1	
	A	00		0	0	
	B	2A		FIM	5	
	C	00		0	0	
	D	20	SVAL	FIM	0	CLEAR Q & x REGISTERS
	E	00		0	0	
	F	A0		LD	0	
	30	BD		XCH	D	
	1	AF		LD	F	
	2	F1	A3	CLC		CALCULATE Q, x, Q'
	3	F6		RAR		
	4	1A		JCN	C0	
	5	39			A4	
	6	6D		INC	D	
	7	40		JUN		
	8	32			A3	
	9	B1	A4	XCH	1	
	A	51		JMS		
	B	0F			EIGHT	
	C	AC		LD	C	
	D	81		ADD	1	
	E	B1		XCH	1	
	F	AD		LD	D	

INDEX REGISTERS

E	CAL/SAL COUNTER	P7	COEFF. COUNTER
C	X	P6	x
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0	Q'	P0	

PROGRAM ASSEMBLY FORM

TITLE

DATE

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND	COMMENTS
0	40	F4		CMA		
	1	B7		XCH	7	
	2	28		FIM	4	CALCULATE S
	3	01		0	1	
	4	77	A5	ISZ	7	
	5	48			A6	
	6	40		JUN		
	7	50			A7	
	8	A9	A6	LD	9	
	9	F5		RAL		
	A	B9		XCH	9	
	B	A8		LD	8	
	C	F5		RAL		
	D	B8		XCH	8	
	E	40		JUN		
	F	44			A5	
	50	AB	A7	LD	B	SAVE SIGN OF
	1	B7		XCH	7	ACCUMULATED PRODUCTS
	2	AC		LD	C	
	3	F1		CLC		
	4	F5		RAL		CHECKING
	5	12		JCN	C1	FOR
	6	91			A12	X >/ 8
	7	F2		IAC		
	8	B6		XCH	6	
	9	6C		INC	C	
	A	2A		FIM	5	
	B	00		0	0	
	C	51		JMS		
	D	1C			MULT	
	E	AB		LD	B	
	F	F8		DAC		

INDEX REGISTERS

E		P7	
C		P6	
A		P5	S
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

PROGRAM ASSEMBLY FORM

HEXADECIMAL			MNEMONIC			TITLE	DATE
PAGE ADR	LINE ADR	INSTR	LABEL	INSTRUCTION OPERATION	INSTRUCTION OPERAND	COMMENTS	
0	60	BA		XCH	A		INDEX REGISTERS
	1	14		JCN	A0		E P7
	2	65			A 8	CHECKING FOR	C P6
	3	12		JCN	C1	S >/ 16	A — S' — P5 →
	4	91			A 12		8 P4
	5	F1	A8	CLC			6 P3
	6	AA		LD	A		4 P2
	7	F5		RAL		CALCULATE	2 P1
	8	BB		XCH	B	S' = SX2	0 P0
	9	AE		LD	E		
	A	F5		RAL			
	B	BA		XCH	A		
	C	2B		SRC	5		
	D	AE		LD	E		
	E	1C		JCN	A1		
	F	73			A 9		
	70	D5		LDM	5		
	1	40		JUN		READ SIGNS AND	
	2	74			A 10	GENERATE SIGN OF PRODUCT	
	3	D9	A9	LDM	9		
	4	80	A 10	ADD	0		
	5	B0		XCH	0		
	6	52		JMS			
	7	D0			SIGN		
	8	51		JMS			
	9	67			SIGNOUT		
	A	A8		LD	8		
	B	1A		JCN	C0		
	C	7F			A 11		
	D	F6		RAR			
	E	F1		CLC			
	F	89	A 11	ADD	9		

PROGRAM ASSEMBLY FORM

TITLE

DATE

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND	COMMENTS
0	80	B9		XCH	9	
	1	E9		RDM		READ WALSH COEFF.
	2	B6		XCH	6	AND CONVERSION
	3	6B		INC	B	ELEMENT AND MULTIPLY
	4	2B		SRC	5	
	5	A7		LD	7	SAVE SIGN OF
	6	BB		XCH	B	ACCUMULATED PRODUCTS
	7	E9		RDM		
	8	B7		XCH	7	
	9	A9		LD	9	SAVE SIGN OF
	A	BA		XCH	A	PRODUCT
	B	52		JMS		
	C	DB			READ F	
	D	51		JMS		
	E	CA			MULTIPLY	
	F	40		JUN		RETURN TO FORM
	90	2D			SVAL	NEXT PRODUCT
	1	51	A12	JMS		
	2	53			SELECT	STORE SIGN OF
	3	A1		LD	1	FOURIER COEFFICIENT
	4	BD		XCH	D	
	5	51		JMS		
	6	82			SIGNIN	
	7	A7		LD	7	
	8	14		JCN	A0	OBTAIN 2'S COMPLEMENT
	9	AD			A13	OF FOURIER COEFFICIENT,
	A	A3		LD	3	IF IT IS NEGATIVE
	B	F4		CMA		
	C	B3		XCH	3	
	D	A2		LD	2	
	E	F4		CMA		
	F	B2		XCH	2	

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8	WALSH	P4	COEFF.
6	F 64	P3	COEFF.
4	PRO-	P2	DUCT
2	PRO-	P1	DUCT
0		P0	

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
0	A0	A5		LD	5
	1	F4		CMA	
	2	B5		XCH	5
	3	A4		LD	4
	4	F4		CMA	
	5	B4		XCH	4
	6	73		ISZ	3
	7	AD			A 13
	8	72		ISZ	2
	9	AD			A 13
	A	75		ISZ	5
	B	AD			A 13
	C	64		INC	4
	D	20	A 13	FIM	0
	E	40		4	0
	F	AF		LD	F
	B0	B1		XCH	1
	1	52		JMS	
	2	DF			FETCH K
	3	2C		FIM	6
	4	00		0	0
	5	2A		FIM	5
	6	00		0	0
	7	28		FIM	4
	8	00		0	0
	9	20		FIM	0
	A	00		0	0
	B	51		JMS	
	C	46			MUL
	D	AA		LD	A
	E	F5		RAL	
	F	BD		XCH	D

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7	
C	COMPEN-	P6	SATED
A	FOURIER	P5	
8	COEFFI-	P4	CIENT.
6	K MATRIX	P3	ELEMENT
4	UNCOMP-	P2	ENSATED
2	FOURIER	P1	COEFF.
0		P0	

COMPENSATE FOR TRUNCATION

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND	COMMENTS
0	C0	F5		RAL		
	1	BC		XCH	C	
	2	F5		RAL		
	3	BD		XCH	D	SELECT 8 BITS
	4	F5		RAL		(SHOWN BOXED)
	5	1A		JCN	C0	AND ROUND OFF
	6	CA			A 14	
	7	7C		ISZ	C	
	8	CA			A 14	
	9	6D		INC	D	
	A	AD	A 14	LD	D	
	B	E0		WRM		
	C	51		JMS		STORE FOURIER
	D	53			SELECT	COEFFICIENT IN RAM.
	E	61		INC	1	
	F	21		SRC	0	
	D0	AC		LD	C	
	1	E0		WRM		
	2	51		JMS		
	3	5E			CHECK	ALL COEFFICIENTS COMPUTED ?
	4	14		JCN	A0	
	5	25			CONVERT	NO
	6	2E		FIM	7	YES
	7	00		0	0	
	8	22		FIM	1	
	9	10		1	0	SELECT OUTPUT PORTS
	A	24		FIM	2	
	B	20		2	0	
	C	26		FIM	3	
	D	00		0	0	
	E	28		FIM	4	
	F	30		3	0	

TITLE

DATE

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND	TITLE / COMMENTS
0	E0	29		SRC	4	SET FLAG
	1	E2		WRR		"COMPUTATION COMPLETE"
	2	51	STAR	JMS		
	3	53			SELECT	
	4	E9		RDM		
	5	23		SRC	1	
	6	E2		WRR		
	7	61		INC	1	
	8	21		SRC	0	
	9	AF		LD	F	OUTPUT
	A	F5		RAL		COEFFICIENTS
	B	BB		XCH	B	
	C	51		JMS		
	D	67			SIGN OUT	
	E	E9		RDM		
	F	25		SRC	2	
	F0	E2		WRR		
	1	A8		LD	8	
	2	1A		JCN	C0	
	3	F8			A 15	
	4	F1		CLC		
	5	F6		RAR		
	6	40		JUN		
	7	FB			A 16	
	8	F6	A 15	RAR		
	9	A3		LD	3	
	A	F5		RAL		
	B	27	A 16	SRC	3	
	C	E2		WRR		
	D	19	A 17	JCN	T1	WAIT FOR "TEST" = "1"
	E	FD			A 17	
	F	51		JMS		

DATE

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
1	00	5E			CHECK
	1	1C	A18	JCN	A1
	2	01			A18
	3	11	A19	JCN	T0
	4	03			A19
	5	40		JUN	
	6	E2			STAR
	7	F1	89←	CLC	
	8	A9		LD	9
	9	F5		RAL	
	A	B9		XCH	9
	B	A8		LD	8
	C	F5		RAL	
	D	B8		XCH	8
	E	C0		BBL	0
	F	28	EIGHT	FIM	4
	10	0D		0	D
	1	A1	EA1	LD	1
	2	F5		RAL	
	3	B1		XCH	1
	4	A0		LD	0
	5	F5		RAL	
	6	B0		XCH	0
	7	79		ISZ	9
	8	11			EA1
	9	C0		BBL	0
	A	51	NEXT BIT 1	JMS	
	B	07			89←
	C	53	MULT	JMS	
	D	44			6→
	E	1A		JCN	C0
	F	22			ZERO1

TITLE

DATE

COMMENTS

ALL COEFFICIENTS OUTPUTTED ?
YES. HALT
NO. WAIT FOR "TEST" = "0"
RETURN TO OUTPUT
NEXT COEFFICIENT

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

89←
(SHIFT R8, R9 LEFT 1 BIT)

SUBROUTINES

EIGHT
(MULTIPLY Q BY 8)

MULT
(MULTIPLY R6 BY R8, R9)

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
1	20	51		JMS	
	1	26			ADD1
	2	A6	ZERO1	LD	6
	3	1C		JCN	A1
	4	1A			NEXT BIT 1
	5	C0		BBL	0
	6	F1	ADD1	CLC	
	7	A9		LD	9
	8	8B		ADD	B
	9	BB		XCH	B
	A	A8		LD	8
	B	8A		ADD	A
	C	BA		XCH	A
	D	C0		BBL	0
	E	F1	67→	CLC	
	F	A6		LD	6
	30	F6		RAR	
	1	B6		XCH	6
	2	A7		LD	7
	3	F6		RAR	
	4	B7		XCH	7
	5	C0		BBL	0
	6	F1	0189←	CLC	
	7	A9		LD	9
	8	F5		RAL	
	9	B9		XCH	9
	A	A8		LD	8
	B	F5		RAL	
	C	B8		XCH	8
	D	A1		LD	1
	E	F5		RAL	
	F	B1		XCH	1

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

ADD1
(ADD P4 AND P5)

67→
(SHIFT R6 AND R7
RIGHT 1 BIT)

0189←
(SHIFT R0, R1, R8 AND R9
LEFT 1 BIT)

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
1	40	A0		LD	0
	1	F5		RAL	
	2	B0		XCH	0
	3	C0		BBL	0
	4	53	NEXT BIT	JMS	
	5	49			01 45 23 ←
	6	51	MUL	JMS	
	7	2E			67 →
	8	1A		JCN	C0
	9	4C			ZERO
	A	53		JMS	
	B	5D			ADD
	C	A7	ZERO	LD	7
	D	1C		JCN	A1
	E	44			NEXT BIT
	F	A6		LD	6
	50	1C		JCN	A1
	1	44			NEXT BIT
	2	C0		BBL	0
	3	F1	SELECT	CLC	
	4	AF		LD	F
	5	F5		RAL	
	6	B1		XCH	1
	7	AE		LD	E
	8	F5		RAL	
	9	F2		IAC	
	A	F2		IAC	
	B	B0		XCH	0
	C	21		SRC	0
	D	C0		BBL	0
	E	7F	CHECK	ISZ	F
	F	[illegible]			CHA 1

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

MUL
(MULTIPLY P3 BY P1 AND P2)

SELECT
(SELECT RAM LOCATION FOR FOURIER COEFFICIENT)

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
1	60	AE		LD	E
	1	1C		JCN	A1
	2	66			CHA2
	3	2E		FIM	7
	4	20		2	0
	5	C0	CHA1	BBL	0
	6	C1	CHA2	BBL	1
	7	F1	SIGNOUT	CLC	
	8	AB		LD	B
	9	F6		RAR	
	A	F6		RAR	
	B	14		JCN	A0
	C	7C			S01
	D	F8		DAC	
	E	14		JCN	A0
	F	79			S02
	70	F8		DAC	
	1	14		JCN	A0
	2	76			S03
	3	EF		RD3	
	4	41		JUN	
	5	7D			S04
	6	EE	S03	RD2	
	7	41		JUN	
	8	7D			S04
	9	ED	S02	RD1	
	A	41		JUN	
	B	7D			S04
	C	EC	S01	RD0	
	D	B8	S04	XCH	8
	E	AB		LD	B
	F	F6		RAR	

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

CHECK
(CHECK FOR PROCESSING OF ALL COEFFICIENTS)

SIGNOUT
(READ SIGN FROM RAM)

87

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
1	80	F6		RAR	
	1	C0		BBL	0
	2	F1	SIGNIN	CLC	
	3	AD		LD	D
	4	F6		RAR	
	5	F1		CLC	
	6	F6		RAR	
	7	14		JCN	A0
	8	9E			SI1
	9	F8		DAC	
	A	14		JCN	A0
	B	AB			SI2
	C	F8		DAC	
	D	14		JCN	A0
	E	BA			SI3
	F	AF		LD	F
	90	F6		RAR	
	1	1A		JCN	C0
	2	9A			SI4
	3	EF		RD3	
	4	F6		RAR	
	5	A7		LD	7
	6	F5		RAL	
	7	E7		WR3	
	8	41		JUN	
	9	C7			SI8
	A	A7	SI4	LD	7
	B	E7		WR3	
	C	41		JUN	
	D	C7			SI8
	E	1A	SI1	JCN	C0
	F	A7			SI5

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

SIGNIN
(WRITE SIGN INTO RAM)

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
1	A 0	EC		RD 0	
	1	F6		RAR	
	2	A7		LD	7
	3	F5		RAL	
	4	E4		WR 0	
	5	41		JUN	
	6	C7			SI 8
	7	A7	SI 5	LD	7
	8	E4		WR 0	
	9	41		JUN	
	A	C7			SI 8
	B	AF	SI 2	LD	F
	C	F6		RAR	
	D	1A		JCN	C0
	E	B6			SI 6
	F	ED		RD 1	
	B 0	F6		RAR	
	1	A7		LD	7
	2	F5		RAL	
	3	E5		WR 1	
	4	41		JUN	
	5	C7			SI 8
	6	A7	SI 6	LD	7
	7	E5		WR 1	
	8	41		JUN	
	9	C7			SI 8
	A	AF	SI 3	LD	F
	B	F6		RAR	
	C	1A		JCN	C0
	D	C5			SI 7
	E	EE		RD 2	
	F	F6		RAR	

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

SIGN IN
(CONT'D)

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
1	C0	A7		LD	7
	1	F5		RAL	
	2	E6		WR2	
	3	41		JUN	
	4	C7			SI8
	5	A7	SI7	LD	7
	6	E6		WR2	
	7	C0	SI8	BBL	0
	8	51	NEXT BIT 2	JMS	
	9	36			0189 ←
	A	51	MULTIPLY	JMS	
	B	2E			67 →
	C	1A		JCN	C0
	D	D8			ZERO 2
	E	AA		LD	A
	F	F6		RAR	
	D0	1A		JCN	C0
	1	D6			MUL1
	2	52		JMS	
	3	E1			SUB2
	4	41		JUN	
	5	D8			ZERO2
	6	51	MUL1	JMS	
	7	DF			ADD2
	8	A7	ZERO 2	LD	7
	9	1C		JCN	A1
	A	C8			NEXT BIT 2
	B	A6		LD	6
	C	1C		JCN	A1
	D	C8			NEXT BIT 2
	E	C0		BBL	0
	F	F1	ADD 2	CLC	

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

MULTIPLY

(MULTIPLY R6,R7 BY R8,R9)

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
1	E 0	A9		LD	9
	1	83		ADD	3
	2	B3		XCH	3
	3	A8		LD	8
	4	82		ADD	2
	5	B2		XCH	2
	6	A1		LD	1
	7	85		ADD	5
	8	B5		XCH	5
	9	A0		LD	0
	A	84		ADD	4
	B	B4		XCH	4
	C	1A		JCN	C0
	D	F0			AD1
	E	F0		CLB	
	F	BB		XCH	B
	F 0	C0	AD1	BBL	0
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				
	9				
	A				
	B				
	C				
	D				
	E				
	F				

TITLE

DATE

COMMENTS

ADD 2
(ADD P0 AND P3
TO P2 AND P1)

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	LINE ADR	INSTR	MNEMONIC LABEL	INSTRUCTION OPERATION	OPERAND	TITLE / COMMENTS
2	00	A3				
	1	44				
	2	0D				
	3	20				Row 0
	4	03			THE	
	5	01				
	6	07			F 64	
	7	10				
	8	36			MATRIX	
	9	83				
	A	58				Row 1
	B	24				
	C	0B				
	D	1B				
	E	28				
	F	10				
	10	21				
	1	4F				
	2	76				
	3	31				Row 2
	4	1A				
	5	3F				
	6	2A				
	7	11				
	8	17				
	9	0A				
	A	30				
	B	75				Row 3
	C	60				
	D	28				
	E	08				
	F	13				

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
2	20	12			
	1	08			
	2	26			
	3	5B			
	4	6F			
	5	2E			
	6	09			
	7	16			
	8	0F			
	9	24			
	A	36			
	B	16			
	C	29			
	D	64			
	E	43			
	F	1C			
	30	0D			
	1	1E			
	2	14			
	3	08			
	4	1C			
	5	43			
	6	64			
	7	29			
	8	0B			
	9	05			
	A	01			
	B	02			
	C	16			
	D	09			
	E	2E			
	F	CE			

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

Row 4

Row 5

Row 6

Row 7

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
2	40	40			
	1	41			
	2	42			
	3	43			
	4	45			THE
	5	48			
	6	4B			K
	7	4F			
	8	54			MATRIX
	9	59			
	A	60			
	B	68			
	C	72			
	D	7E			
	E	8C			
	F	4F			
	50	0			
	1	0			
	2	1			
	3	0			
	4	1			
	5	1			
	6	1			
	7	0			
	8	1			
	9	0			
	A	0			
	B	0			
	C	0			
	D	1			
	E	0			
	F	0			

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

SIGNS FOR
CAL/COSINE CONVERSION

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
2	60	0			
	1	1			
	2	0			
	3	0			
	4	1			
	5	0			
	6	0			
	7	0			
	8	1			
	9	1			
	A	1			
	B	0			
	C	0			
	D	0			
	E	1			
	F	0			
	70	0			
	1	0			
	2	0			
	3	1			
	4	0			
	5	0			
	6	1			
	7	0			
	8	1			
	9	0			
	A	1			
	B	1			
	C	1			
	D	0			
	E	0			
	F	0			

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7		F
C		P6		D
A		P5		B
8		P4		9
6		P3		7
4		P2		5
2		P1		3
0		P0		1

SIGNS FOR
CAL/COSINE CONVERSION
(CONT'D)

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
2	80	0			
	1	1			
	2	1			
	3	1			
	4	0			
	5	1			
	6	0			
	7	0			
	8	1			
	9	1			
	A	0			
	B	1			
	C	1			
	D	1			
	E	1			
	F	0			
	90	0			
	1	1			
	2	1			
	3	1			
	4	1			
	5	0			
	6	1			
	7	1			
	8	0			
	9	0			
	A	1			
	B	0			
	C	1			
	D	1			
	E	1			
	F	0			

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

SIGNS FOR
SAL/SINE CONVERSION

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
2	A 0	0			
	1	0			
	2	0			
	3	1			
	4	1			
	5	1			
	6	0			
	7	1			
	8	0			
	9	1			
	A	0			
	B	0			
	C	1			
	D	0			
	E	0			
	F	0			
	B 0	0			
	1	1			
	2	0			
	3	0			
	4	0			
	5	1			
	6	1			
	7	1			
	8	0			
	9	0			
	A	0			
	B	1			
	C	0			
	D	0			
	E	1			
	F	0			

TITLE

DATE

COMMENTS

INDEX REGISTERS			
E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

SIGNS FOR
SAL/SINE CONVERSION
(CONT'D)

PROGRAM ASSEMBLY FORM

TITLE

DATE

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND	COMMENTS
2	C0	0				INDEX REGISTERS
	1	0				E ____ P7 ____
	2	1				C ____ P6 ____
	3	0				A ____ P5 ____
	4	0				8 ____ P4 ____
	5	0				6 ____ P3 ____
	6	0				4 ____ P2 ____
	7	1				2 ____ P1 ____
	8	0				0 ____ P0 ____
	9	1				
	A	1				
	B	1				
	C	0				
	D	1				
	E	0				
	F	0				
	D0	38	SIGN	FIN	4	
	1	AE		LD	E	
	2	1C		JCN	A1	SIGN
	3	D7			SA1	(READ SIGN OF
	4	DB		LDM	11	CONVERSION ELEMENT)
	5	42		JUN		
	6	D8			SA2	
	7	D7	SA1	LDM	7	
	8	80	SA2	ADD	0	
	9	B0		XCH	0	
	A	C0		BBL	0	
	B	38	READF	FIN	4	READ F
	C	20		FIM	0	(READ CONVERSION ELEMENT)
	D	00		0	0	
	E	C0		BBL	0	
	F	36	FETCH K	FIN	3	FETCH K

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
2	E 0	C0		BBL	0
	1	F1	SUB 2	CLC	
	2	A3		LD	3
	3	99		SUB	9
	4	F3		CMC	
	5	B3		XCH	3
	6	A2		LD	2
	7	98		SUB	8
	8	B2		XCH	2
	9	F3		CMC	
	A	A5		LD	5
	B	91		SUB	1
	C	B5		XCH	5
	D	F3		CMC	
	E	A4		LD	5
	F	90		SUB	0
	F 0	B4		XCH	4
	1	12		JCN	C1
	2	F5			SU1
	3	D1		LDM	1
	4	BB		XCH	B
	5	C0	SU1	BBL	0
	6				
	7				
	8				
	9				
	A				
	B				
	C				
	D				
	E				
	F				

TITLE

DATE

COMMENTS

(READ K ELEMENT)

SUB 2
(SUBTRACT P0 & P4 FROM P2 & P1)

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

99.

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND	TITLE / COMMENTS
3	00	51				
	1	00				
	2	22				
	3	00				
	4	07				
	5	00				
	6	10			WALSH	CAL
	7	00			SPECTRUM	COEFFICIENTS
	8	02			OF	
	9	00			f(t) = cos t +	
	A	01			0.25 sin t	
	B	00			− 1.25 sin 2t	
	C	03			+ 1.00 sin 3t	
	D	00				
	E	08				
	F	00				
	10	30				
	1	66				
	2	39				
	3	00				
	4	2D				
	5	2A				SAL
	6	0E				COEFFICIENTS.
	7	00				
	8	06				
	9	08				
	A	0D				
	B	00				
	C	15				
	D	14				
	E	06				
	F	00				

DATE

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

100.

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	LINE ADR	INSTR	MNEMONIC LABEL	INSTRUCTION OPERATION	INSTRUCTION OPERAND
3	20	0			
	1	0			
	2	0			
	3	0			
	4	1			
	5	0			
	6	0			
	7	0			
	8	1			
	9	0			
	A	1			
	B	0			
	C	1			
	D	0			
	E	0			
	F	0			
	30	0			
	1	1			
	2	0			
	3	0			
	4	1			
	5	0			
	6	0			
	7	0			
	8	1			
	9	0			
	A	1			
	B	0			
	C	1			
	D	0			
	E	0			
	F	0			

TITLE

DATE

COMMENTS

CAL
COEFFICIENT SIGNS

INDEX REGISTERS			
E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

SAL
COEFFICIENT SIGNS.

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
3	40	3A	WALSH	FIN	5
	1	C0		BBL	0
	2	36	WALSIGN	FIN	3
	3	C0		BBL	0
	4	F1	6→	CLC	
	5	A6		LD	6
	6	F6		RAR	
	7	B6		XCH	6
	8	C0		BBL	0
	9	F1	014523←	CLC	
	A	A3		LD	3
	B	F5		RAL	
	C	B3		XCH	3
	D	A2		LD	2
	E	F5		RAL	
	F	B2		XCH	2
	50	A5		LD	5
	1	F5		RAL	
	2	B5		XCH	5
	3	A4		LD	4
	4	F5		RAL	
	5	B4		XCH	4
	6	A1		LD	1
	7	F5		RAL	
	8	B1		XCH	1
	9	A0		LD	0
	A	F5		RAL	
	B	B0		XCH	0
	C	C0		BBL	0
	D	F1	ADD	CLC	
	E	A3		LD	3
	F	39		ADD	9

TITLE

DATE

COMMENTS

WALSH

(READ WALSH COEFFICIENT)

WALSIGN

(READ SIGN OF WALSH COEFF)

6→

(SHIFT R6 RIGHT 1 BIT)

014523←

(SHIFT R0, R1, R4, R5, R2, R3 LEFT 1 BIT)

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

PROGRAM ASSEMBLY FORM

HEXADECIMAL PAGE ADR	HEXADECIMAL LINE ADR	HEXADECIMAL INSTR	MNEMONIC LABEL	MNEMONIC INSTRUCTION OPERATION	MNEMONIC INSTRUCTION OPERAND
3	60	B9		XCH	9
	1	A2		LD	2
	2	88		ADD	8
	3	B8		XCH	8
	4	A5		LD	5
	5	8B		ADD	B
	6	BB		XCH	B
	7	A4		LD	4
	8	8A		ADD	A
	9	BA		XCH	A
	A	A1		LD	1
	B	8D		ADD	D
	C	BD		XCH	D
	D	A0		LD	0
	E	8C		ADD	C
	F	BC		XCH	C
	70	C0		BBL	0
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				
	9				
	A				
	B				
	C				
	D				
	E				
	F				

TITLE

DATE

COMMENTS

INDEX REGISTERS

E		P7	
C		P6	
A		P5	
8		P4	
6		P3	
4		P2	
2		P1	
0		P0	

ADD
(ADD P0, P2, P1
TO P6, P5, P4)

BIBLIOGRAPHY

[1] K.H. Siemens, "Walsh Spectral Analysis", Ph.D. Thesis, McMaster University, Hamilton, Ontario, Canada, 1972.

[2] W.M. Doran, "The Design of a Digital Walsh-Fourier Converter", M.Eng. Thesis, McMaster University, Hamilton, Ontario, Canada, 1972.

[3] C. Cardot, "Définition analytique simple des fonction de Walsh et application à la détermination exacte de leurs propriétés spectrales", Ann. Télécomm., vol. 72, pp. 31-47, Jan.-Feb. 1972.

[4] K.H. Siemens and R. Kitai, "Walsh Series to Fourier Series conversion", Proc. Symp. Appl. of Walsh Functions, Washington,1972, pp. 295-297.

[5] E.A. Torrero, "Focus on Microprocessors", Electronic Design, vol. 22, no. 18, pp. 52-69, 1974.

[6] "Intel MCS-40 User's Manual for Logic Designers", Intel Corporation, Santa Clara, Calif., 1974.