THE CORDIC ALGORITHM IMPLEMENTATION FOR TRIGONOMETRIC FUNCTION EVALUATION IN HP21MX

By

PEIHSUNG THOMAS HU

Bachelor of Science

National Chiao Tung University

Hsinchu, Taiwan

1972

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
MASTER OF SCIENCE
May, 1978

Thesis
1978
H874c
cop.2

OKLAHOMA STATE
UNIVERSITY
LIBRARY

THE CORDIC ALGORITHM IMPLEMENTATION FOR

TRIGONOMETRIC FUNCTION EVALUATION

IN HP21MX

Thesis Approved:

D. E. Hedrick
Thesis Adviser

T. E. Bailey

E. L. Shreve

Norman N. Durham
Dean of the Graduate College

1009257

PREFACE

This paper describes the Cordic algorithm and its implementation for the evaluation of the sine function in a HP21MX computer. A polynomial method is also described and implemented in the HP21MX computer for the purpose of comparing the result with the the Cordic algorithm. The HP21MX microprogramming is also applied in this experiment to increase the programming efficiency.

I would like to express my gratitude to my major advisers, Dr. Edward Shreve and Dr. G.E. Hedrick for their advice and guidance during this project. Also, appreciation is expressed to my other committee member, Dr. T.E. Bailey for his invaluable assistance in the preparation of the final manuscript. Thanks are also extended to Mrs. Pam Haught for her typing this paper and her invaluable help in preparing the final copy of this paper.

TABLE OF CONTENTS

Chapter | Page

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Symbol	Dimension	Function
ADC		Address computation defined operation
EXEC		Instruction execution defined operation
IOIG		I/O interrupt generator system program
MAC		Memory access defined operation
PROC		Processor unit system program
RUN		Run indicator
A	16	Accumulator (See Chapter IV)
B	16	Accumulator extension
C	16	Local vector
D		Decoding matrices
E	1	Extend register
F	56	I/O device flag
M	2^{15}, 16	Main memory
N	167, 9	Navigation matrix (See Figure 4)
I	16	Instruction register
O	1	Overflow register
P	16	Program counter
Q	12	Mask vector
U	12	OP code vector
S		Current interrupt priority level
T	16	T-bus

V	56	I/O device control bit
X	16	X-register
Y	16	Y-register
Z	56,8	I/O device data buffer
a,b,m,t,i,j		Local vectors
d	2	Local vectors
e	4	Program exceptions
e_0		Power fail
e_1		Memory parity
e_2		Dual-channel port controller 1
e_3		Dual-channel port controller 2
g	16	Local vectors
h	2	Interrupt holder
h_0		Exceptions
h_1		I/O interrupt
l	16	Local Vector
n	9	Navigation vector
n_0,n_1,n_3		Branch control in EXEC
n_2		Entry line in EXEC
n_4		Instruction class
q	4	Memory access quene
r	4	Memory access request
v	9	Temporary navigation vector

CHAPTER I

INTRODUCTION

In the past, the transcendental functions were computed by mathematicians using many different algorithms. Power series, polynomial expansions, continued fractions, and Chebyshev polynomials have all been used. Since the advent of large scale computing in the twentieth century, many mathematical functions including trancendental functions have been calculated by computers. As a general rule, multiplication and division are very time-consuming functions compared to addition and subtraction implemented in a computer. A review of the conventional methods which are used for solving transcendental functions, such as power series, polynomial expansions, continued fractions, and Chebyshev polynomials, shows that a number of multiplications and divisions are required that results in inefficiency of implementation.

Therefore, much effort has been made to search for alternate ways which can best suit the requirements of speed and programming efficiency for real-time applications.

Henry Briggs (17) first developed the concept of pseudo-division and pseudo-multiplication in 1924. He used this method to generate a table of logarithms.

In 1959, J. E. Volder (9) described a Coordinate Rotation Digital Computer (Cordic) for the calculation of trigonometric functions, multiplication, division, and conversion between binary and mixed radix number systems. In the same year, Dagget (10) discussed the use of the Cordic computer for decimal-binary conversion. In 1962, Meggitt (11) developed a pseudo-division and pseudo-multiplication processor using the Cordic technique, while in 1971 J. S. Walther (12) developed a technique for calculating elementary functions using Cordic. David S. Cochran (14) in 1972 implemented the Cordic algorithm in HP 35 calculators, and Despain (13) in 1974 developed a technique for Fourier transformation using the Cordic algorithm.

Generally speaking, the trigonometric functions are calculated by polynomial expansions, power series, or Chebyshev polynomials in most current general purpose computers.

The major goal of this thesis is to implement the Cordic algorithm in a general purpose computer for evaluation of trigonometric functions. The speed and accuracy of the results are observed and compared with those of conventional algorithms. Microprogramming has been used in this research to increase the program efficiency. The anticipated result is to determine the best way of evaluating the trigonometric functions, which can reduce the computer execution time to a minimum and give reasonable accuracy of the results.

Only the sine function is implemented as a part of this research. The tasks are divided into four parts:

1. Implement the Cordic algorithm in an assembly coded program.
2. Implement the Cordic algorithm in a microprogram.
3. Implement one of the conventional methods in an assembly coded program.
4. Implement the same conventional method in a microprogram.

CHAPTER II

STANDARD TECHNIQUE FOR THE EVALUATION OF TRIGONOMETRIC FUNCTIONS

The evaluation of elementary functions for various values of their arguments is required to solve a number of mathematical problems. Because of this, the computation of values of elementary functions was an important factor in stimulating the development of mathematical analysis. Therefore, a great deal of effort has been made by many mathematicians in the past two centuries to find methods of evaluating these elementary functions. Power series have been and still are used for this purpose. Mercator used a power series for logarithms; Newton used it then for trigonometric and inverse trigonometric functions; and Euler used one for the exponential function. Iterative processes (e.g., Newton's method) were also applied for solving equations (3). Furthermore, in the eighteenth century, many mathematicians (Lambert, Euler, Lagrange, et al.) used continued fractions to represent elementary functions. In recent years the technique of expansions in orthogonal polynomials has been widely applied for computing elementary functions. The Chebyshev polynomials which give good convergence are widely used for this purpose too.

All those methods mentioned above are well documented and are described in many mathematics books; thus it is not necessary to explain them here. Power series for evaluating trigonometric functions are used in this paper as a conventional method of evaluating trigonometric functions in order to compare them to evaluations using the Cordic algorithm. Therefore, for convenience, the power series method is described as follows:

Power Series

The elementary functions can be represented as power series in a number of ways. Consider the Taylor-Maclaurin Series for a given function $f(x)$:

$$f(x) = \sum_{k=0}^{a} \frac{f^{(k)}(0)}{k\,!} x^k \tag{2.1}$$

Truncating this at the nth term produces an nth-degree polynomial $S_n(x)$ (a finite Taylor Series).

$$S_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(0)}{k\,!} x^k \tag{2.2}$$

The polynomial $S_n(x)$ has the following properties:

$$f(x) = S_n(x) + O(x^n) \tag{2.3}$$

where $S_n(x)$ is the unique nth-degree polynomial of best approximation $P_n(x)$, for which

$$f(x) - P_n(x) = O(x^n) \tag{2.4}$$

If $f(x) = \sin(x)$, then $\sin(x)$ can be represented in a power series as:

$$\sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^2k+1}{(2k+1)!} \tag{2.5}$$

Cos(x) can be represented in a power series as:

$$\cos(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!} \tag{2.6}$$

In order to implement this algorithm in a computer for evaluation of trigonometric functions, the number of terms (i.e., constant k) required for specific accuracy is determined first.

To determine the constant k, the maximum accuracy of evaluation in the computer must be known first. The computer used in this research is an HP21MX, the memory word of which contains 16 bits. Although multiple precision could be achieved by using multiple words in arithmetic operations, single precision (single word) is still used in the Cordic algorithm and power series here for the sake of simplicity of programming.

Hastings (4) set up three equations by using power series to evaluate the sine function, which are as follows:

$$\sin \frac{\Pi}{2} x = c_1 x + c_3 x^3 + c_5 x^5 \qquad (2.7)$$

$$c_1 = 1.5706268$$

$$c_3 = -0.6432392$$

$$c_5 = 0.0727102$$

$$\sin \frac{\Pi}{2} x = c_1 x + c_3 x^3 + c_5 x^5 + c_7 x^7 \qquad (2.8)$$

$$c_1 = 1.570794852$$

$$c_3 = -0.645920978$$

$$c_5 = 0.079487663$$

$$c_7 = -0.004362476$$

$$\sin \frac{\Pi}{2} x = c_1 x + c_3 x^3 + c_5 x^5 + c_7 x^7 + c_9 x^9 \qquad (2.9)$$

$$c_1 = 1.57079631847$$

$$c_3 = -0.64596371100$$

$$c_5 = 0.07968967928$$

$$c_7 = -0.00467376557$$

$$c_9 = 0.00015148419$$

where $-1 \le x \le 1$

To determine which equation will be used in this paper, the maximum value of the error of each equation is checked. The maximum value of the error is 0.0001 for equation (2.7), 0.000001 for equation (2.8), and 0.000000005 for equation (2.9). For a 16-bit computer word, the maximum accuracy that can be represented is 5 decimal digits.

The accuracy of equations (2.8) and (2.9) is more than 5 decimal digits. If they are used to evaluate sine functions in a 16-bit word machine, they will consume a lot more execution time than equation (2.7) with just a slightly more accurate result. Therefore, in order to get the best execution time and accuracy, equation (2.7) is used in this research.

CHAPTER III

THE CORDIC ALGORITHM

INTRODUCTION

Cordic is a special purpose, binary computer which contains a unique arithmetic unit which differs from the arithmetic unit of conventional computers. Although Cordic is a single processor computer, its arithmetic unit is composed of three shift registers and three adder-subtractors which are operated in parallel instead of sequentially. Each programmed operation is accomplished in a fixed number of steps. Each step involves modifying three numbers which reside in three arithmetic unit registers by adding or subtracting a constant for each one. Setting of all three adder-subtractors is controlled by the sign of the quantity in one of the arithmetic unit registers. In this way, calculations related to the addition or subtraction of constants can be executed simultaneously.

Functional Description

There are two computing modes in Cordic for the trigonometric operations: ROTATION and VECTORING. In the ROTATION mode the coordinate components of a vector and an angle of rotation are given and the coordinate components of the original vector, after rotation through the given angle, are computed. In the VECTORING mode, the coordinate components of a vector are given and the magnitude and angular argument of the original vector are computed. The basic computing technique used in both the ROTATION and VECTORING modes in Cordic is a step-by-step sequence of pseudo-rotations which result in an overall rotation through a given angle (ROTATION) or result in a final angular argument of zero (VECTORING).

It is necessary that the angular increments of rotation be computed in decreasing order (9). In order to evaluate the sine and cosine functions for the angles from -180° to 180°, the magnitude actually chosen for the first increment should be 90°. The expression for a set of coordinate components, X_1 and Y_1, rotated through plus or minus 90° is simply

$$Y_2 = \pm X_1 = R_1 \sin(\theta_1 \pm 90°) \tag{3.1}$$

$$X_2 = \mp Y_1 = R_1 \cos(\theta \pm 90°) \tag{3.2}$$

Where R_1 and θ, are the magnitude and angle of the vector (X_1, Y_1) and X_2 and Y_2 are the coordinates of vector (X_1, Y_1) after rotating 90°.

The first step is unique in that a perfect rotation step is performed. The remaining computing steps can be clarified by examining relationships involved in a typical rotation step which are shown in Figure 1. Consider two given coordinate components, Y_i and X_i, in the plane coordinate system shown. In this discussion, the quantity i is equal to the number of the particular step under consideration. The components Y_i and X_i are associated with the ith step and describe a vector of magnitude R_i at an angle θ_i with respect to the origin according to the relationships.

$$Y_i = R_i \sin\theta \tag{3.3}$$

$$X_i = R_i \cos\theta \tag{3.4}$$

In Figure 1 the angle α_i is the magnitude of rotation associated with each computing step. The general expression for α_i where $i > 1$ is x

$$\alpha_i = \tan^{-1} 2^{-(i-2)} \tag{3.5}$$

The reason for choosing this particular magnitude of α_i is that a rotation of coordinate components through $\pm\, \alpha_i$ may be accomplished by the simple process of shifting and adding. The two choices of positive or negative rotation are shown in Figure 1. The general expressions for the rotated components are

$$Y_{i+1} = \sqrt{1+2^{-2(i-2)}}\; R_i \sin(\theta_i \pm \alpha_i)$$

$$= Y_i \pm 2^{-(i-2)} X_i \tag{3.6}$$

and

$$X_{i+1} = \sqrt{1 + 2^{-2(i-2)}}\; R_i \cos(\theta_i \pm \alpha_i)$$

$$= X_i \mp 2^{-(i-2)} Y_i \tag{3.7}$$

Note that the right-hand terms of (3.6) and (3.7) may be obtained by two simultaneous shift-and-add operations, if the angular rotation magnitude is restricated to (3.5). This is the fundamental relationship upon which the Cordic computing technique is based.

The computing action of adding (or subtracting) a shifted value

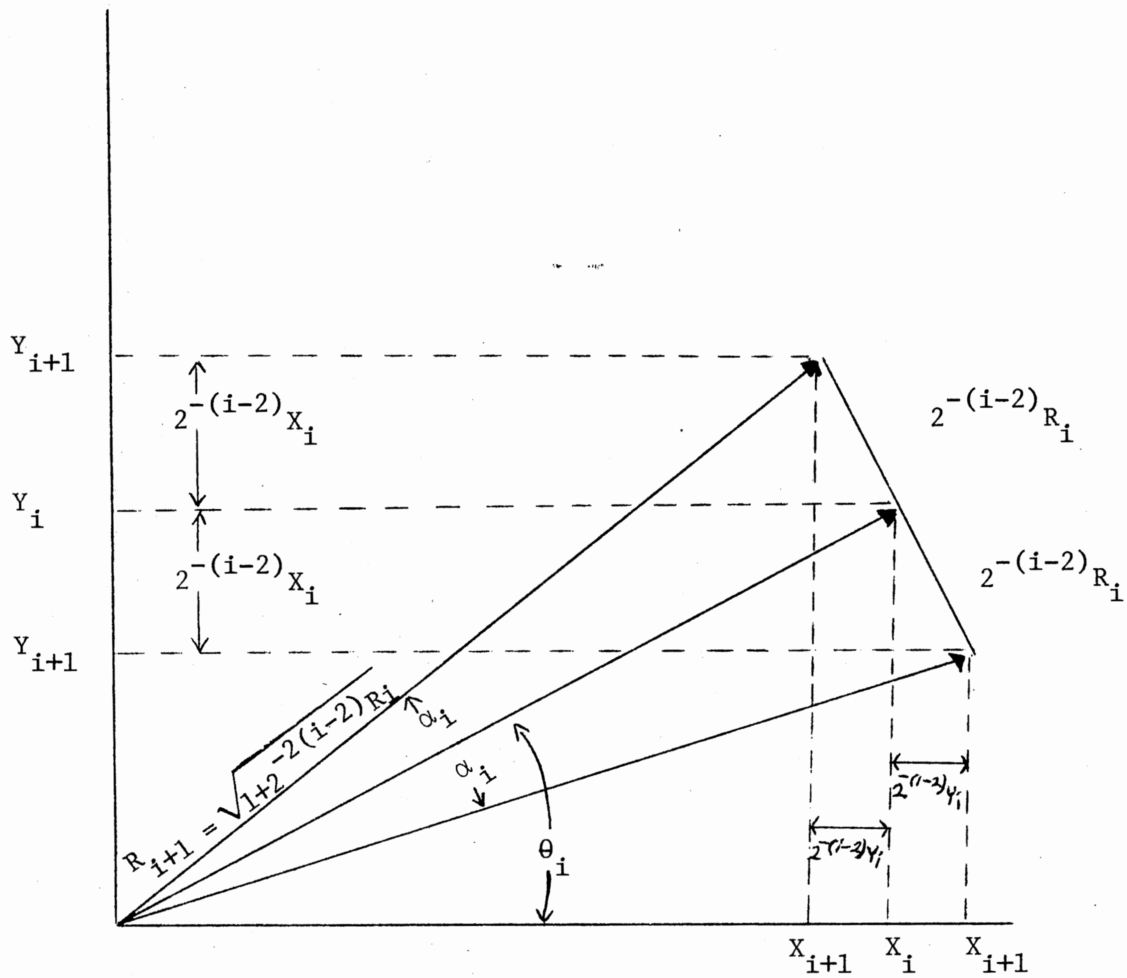

Figure 1. Typical computing step

of X_i to Y_i to obtain Y_{i+1}, while simultaneously subtracting (or adding) a shifted value of Y_i to X_i to obtain X_{i+1} is termed "cross addition".

The terms under the radical in (3.6) and (3.7) indicate the increase in magnitude when $i > 2$; either of the two choices of direction produces the same change in magnitude. If the rotation is always through either a positive or negative α_i at each step, then the increase in magnitude may be considered as a constant. This requirement does not allow the choice of zero rotation at any step. In order to identify the choice in a particular step, the $\pm$ notation may be represented by the binary operator v_i, where v_i can be either +1 or -1. This substition produces the general expressions

$$Y_{i+1} = \sqrt{1 + 2^{-2(i-2)}}\; R_i \sin(\theta_i + v_i\,\alpha_i) \tag{3.8}$$

and

$$X_{i+1} = \sqrt{1 + 2^{-2(i-2)}}\; R_i \cos(\theta_i + v_i\,\alpha_i) \tag{3.9}$$

where $v_i = +1$ or -1

Similarly, after the completion of the rotation step in which the i + 1 terms are obtained, the i + 2 terms may be computed from these terms with the results

$$Y_{i+2} = \sqrt{1 + 2^{-2(i-1)}}\;\sqrt{1 + 2^{-2(i-2)}}\; R_i \sin(\theta_i + v_i\alpha_i + v_{i+1}\alpha_{i+1}) \tag{3.10}$$

and

$$X_{i+2} = \sqrt{1 + 2^{-2(i-1)}}\;\sqrt{1 + 2^{-2(i-2)}}\; R_i \cos(\theta_i + v_i\alpha_i + v_{i+1}\alpha_{i+1}) \tag{3.11}$$

Likewise, these rotation steps can be continued through any finite, pre-determined number of steps. Consider the initial coordinate components Y_1 and X_1 where

$$Y_1 = R_1 \sin\theta \tag{3.12}$$

and

$$X_1 = R_1 \cos\theta \tag{3.13}$$

Suppose the first rotation step is $\pm$ 90° and the number of steps is determined as n. The expressions for the final coordinate components will be

$$Y_{n+1} = (\sqrt{1 + 2^{-0}}\ \sqrt{1 + 2^{-2}}\ \ldots \sqrt{1 + 2^{-2(n-2)}})\ R_1 \sin(\theta_1 + v_1\alpha_1 + v_2\alpha_2 + \ldots + v_n\alpha_n) \tag{3.14}$$

and

$$X_{n+1} = (\sqrt{1 + 2^{-0}}\ \sqrt{1 + 2^{-2}}\ \ldots \sqrt{1 + ^{-2(n-2)}})\ R_1 \cos(\theta_1 + v_1\alpha_1 + v_2\alpha_2 + \ldots + v_n\alpha_n) \tag{3.15}$$

The increase in magnitude of the components for a particular value n is a constant and is represented by k. The value selected for n is a function of the desired computing accuracy and can be a constant for a particular computer. For example,

if $n = 24$,

$k = 1.646760255$.

The basic components required to perform the cross-addition are shown in Figure 2. It has not yet been shown how the prescribed sequence of rotation steps can be controlled to effect the desired over-all rotation. By examination of (3.14) and (3.15), the rotation of a set of coordinate components Y_1 and X_1 through a given angle can be expressed as

$$Y_{n+1} = KR_1 \sin(\theta_1 + \lambda) \tag{3.16}$$

and

$$X_{n+1} = KR_1 \text{Cos}(\theta_1 + \lambda) \tag{3.17}$$

where

$$\lambda = v_1\alpha_1 + v_2\alpha_2 + \ldots + v_n\alpha_n \tag{3.18}$$

In the VECTORING mode,

$$-\theta_1 = \lambda, \text{ ie, } -\theta_1 = v_1\alpha_1 + v_2\alpha_2 + \ldots + v_n\alpha_n \tag{3.19}$$

The sequence of (3.18) and (3.19) form a special radix representation equivalent to the desired angle, λ or θ, where

$$\alpha_1 = 90° \tag{3.20}$$

$$\alpha_2 = \tan^{-1}2^{-0} = 45° \tag{3.21}$$

$$\alpha_3 = \tan^{-1}2^{-1} = 26.5° \tag{3.22}$$

$$\alpha_i = \tan^{-1}2^{-(i-2)} \tag{3.23}$$

The α terms are referred to as ATR (Arctangent Radix) constants and are precomputed and stored in the computer. The v terms are referred to as ATR digits and are determined during each operation.

In the Cordic computer, the ATR digits are determined sequentially, most significant digit first, and are used to control the conditional

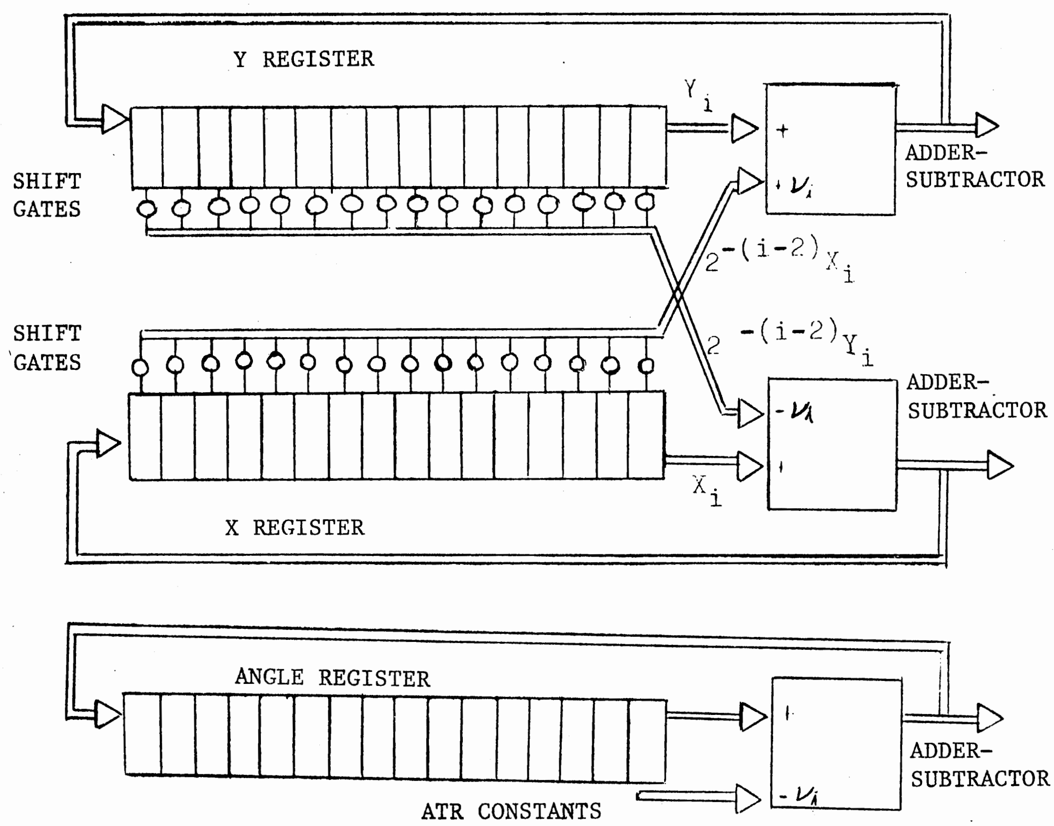

Figure 2. Cordic Arithmetic Unit

action of the adder-subtractors in the arithmetic unit. The following paragraphs contain a description of the manner in which the ATR code representation, ν_1, ν_2, ν_3, ..., ν_n can be determined for any given angle, λ or θ.

First, for any angle λ or θ, there must be at least one set of values of ν for the operators that will satisfy (3.18) and (3.19). Second, a simple technique must be available for determing the ATR code digits that satisfy these equations. The following relationships are necessary and sufficient for any sequence of radix constants to meet the above requirements (3.9).

$$|\lambda \text{ or } \theta| \leq \alpha_1 + \alpha_2 + \alpha_3 + \ldots + \alpha_n + \alpha_n \tag{3.24}$$

$$\alpha_i \leq \alpha_{i+1} + \alpha_{i+2} + \ldots + \alpha_n + \alpha_n \tag{3.25}$$

For the satisfaction of (3.20) through (3.23), it is required that or θ be constrained by

$$-180° \leq \lambda \text{ or } \theta \leq +180° \tag{3.26}$$

Equation (3.26) imposes no special consideration if the two's complement notation is used. By employing an additional register and adder-subtractor (identified in Figure 2 as the angle register) the relationship of (3.16) (ROTATION-mode) can be instrumented by 1) sensing the sign of the angle of rotation (or remainder if $i > 1$) and 2) either subtracting or adding to the angle the ATR constant corresponding to the particular step. In each step, the relationship instrumented is

$$|\lambda_{i+1}| = ||\lambda_i| - \alpha_i| \tag{3.27}$$

Equation (3.24) is equivalent to

$$- \alpha_1 \leq | \lambda | - \alpha_1 \leq \alpha_2 + \alpha_3 + \ldots + \alpha_n + \alpha_n \tag{3.28}$$

Application of the relationships of (3.25) results in

$$| \lambda \quad | \equiv | \, |\lambda_1| - \alpha_1 \, | \leq \alpha_2 + \alpha_3 + \ldots + \alpha_n + \alpha_n \tag{3.29}$$

Continuation of this sequence through α_n results in

$$| \lambda_{n+1} | \leq \alpha_n \tag{3.30}$$

Equation (3.30) can be used to prove that the remainder in the angle register converges to zero in the ROTATION mode (9).

The sequence of operation signs used to null λ to zero is the negative of the equivalent ATR code for the original angle. More simply, the ATR code digit of each step is equal to the sign of the quantity in the angle register before each step. Therefore, simultaneously with each step in the angle register, the ATR code digit may be used to control the cross-addition step in the Y and X registers (shown in Figure 2) to effect a rotation of components through an equal angular increment.

The proof of the convergence of the effective angular argument θ_{n+1} to zero, which is necessary in the VECTORING mode, may be obtained by replacing λ by θ. The sign of the angle θ_i is obtained by sensing the sign of Y_i. The sequence of signs of Y_i is the negative of the ATR code for the effective rotation performed on the components Y_1 and X_1. During each cross-addition operation in the Y and X register, the corresponding ATR constant can be conditionally added or subtracted, depending on v_i, to an accumulating sum in the angle register so that, at the end of the computing sequence, when $\theta_{n+1} = 0$, the quantity in the angle register will be equal to the original angular argument θ_1 of the coordinate components Y_1 and X_1.

The step-by-step results of a typical rotation computing sequence are shown in Table I. The two's complement notation is used for all quantities, and shift quantities are truncated without round-off. The step-by-step results of a typical rotation computing sequence are shown in Table I.

Representation of Angles in Cordic

In Cordic, angles are represented as a binary fraction of a half revolution (Π) with two's complements for negative angles, as shown in Figure 3. Since a one to the left of the binary point is used to represent a negative quantity in the two's complement system, angles from $+180°$ to slightly less than $+360°$ are interpreted internally as negative angles measured clockwise from 0°. For example, 45° in Cordic is

$$\frac{\Pi/4}{\Pi} = \tfrac{1}{4} = (0.25)_{10} = (0.01)_2$$

For 90° the Cordic representation is

$$90° = /2$$

$$\frac{\Pi/2}{\Pi} = \tfrac{1}{2} = (0.5)_{10} = (0.1)_2$$

For 270° the Cordic representation is

$$270° = \frac{3\Pi/2}{\Pi} = (1.5)_{10} = (1.1)_2$$

TABLE I

TYPICAL ROTATION COMPUTING SEQUENCE

Y Register	X Register	Angle Register	
Y_1 = 0.0101110	1.1000101 = X_1	0.1100101 = λ	
+ 1.1000101	- 0.0101110	- 0.1000000	$\tan^{-1} \infty$
1.1000101	1.1010010	0.0100101	
+ 1.1010010	- 1.1000101	- 0.0100000	$\tan^{-1} 1$
1.0010111	0.0001101	0.0000101	
+ 0.0000110	- 1.1001011	- 0.0010010	$\tan^{-1} 2^{-1}$
1.0011101	0.1000010	1.1110011	
- 0.0010000	+ 1.1100111	+ 0.0001001	$\tan^{-1} 2^{-2}$
1.0001101	0.0101001	1.1111100	
- 0.0000101	+ 1.1110001	+ 0.0000101	$\tan^{-1} 2^{-3}$
1.0001000	0.0011010	0.0000001	
+ 0.0000001	- 1.1111000	- 0.0000010	$\tan^{-1} 2^{-4}$
1.0001001	0.0100010	1.1111111	
- 0.0000001	+ 1.1111100	+ 0.0000001	$\tan^{-1} 2^{-5}$
1.00010000	0.0011110	0.0000000	

TABLE II

TYPICAL VECTORING COMPUTING SEQUENCE

Y Register	X Register	Angle Register
Y_1 = 0.0101110	1.1000101 = X_1	0.0000000
− 1.1000101	+ 0.0101110	+ 0.1000000 $\tan^{-1}\infty$
0.0111011	0.0101110	0.1000000
− 0.0101110	+ 0.0111011	+ 0.0100000 $\tan^{-1} 1$
0.0001101	0.1101001	0.1100000
− 0.0110100	+ 0.0000110	+ 0.0010010 $\tan^{-1}2^{-1}$
1.1011001	0.1101111	0.1110010
+ 0.0011011	− 1.1110110	− 0.0001001 $\tan^{-1}2^{-2}$
1.1110100	0.1111001	0.1101001
+ 0.0001111	− 1.1111110	− 0.0000101 $\tan^{-1}2^{-3}$
0.0000011	0.1111011	0.1100100
− 0.0000111	+ 0.0000111	+ 0.0000010 $\tan^{-1}2^{-4}$
1.1111111	0.1111100 = K K_1	0.1100101 = 0

Sine and Cosine Algorithm

As mentioned above, there are two computing modes for Cordic, ROTATION and VECTORING. Evaluating sine or cosine functions makes use of the ROTATION mode by setting the original vector on the X-axis and rotating the vector through an angular argument whose sine or cosine is computed.

Functional Description

In order to use the ROTATION computing sequence (Table I) of Cordic to evaluate sine and cosine functions, several initial conditions and values are set up:

1) The Y-register is initialized with 0.
2) The X-register is initialized with a unit vector.
3) The A-register is initialized with the angle which is going to be computed.
4) A sign digit of 0 in the A-register establishes a v_i of +1, which causes the top adder - subtractor to add, the middle adder-subtractor to subtract, and the bottom adder - subtractor to subtract. A sign digit of 1 has the opposite effect.
5) The number of steps (iterations) is initialized depending on the desired accuracy.

The Cordic ROTATION computing sequence is started as shown in Table 1.

The final result is in the Y-register if the function evaluated is sine and in the X-register if the function evaluated is cosine after the final computation step.

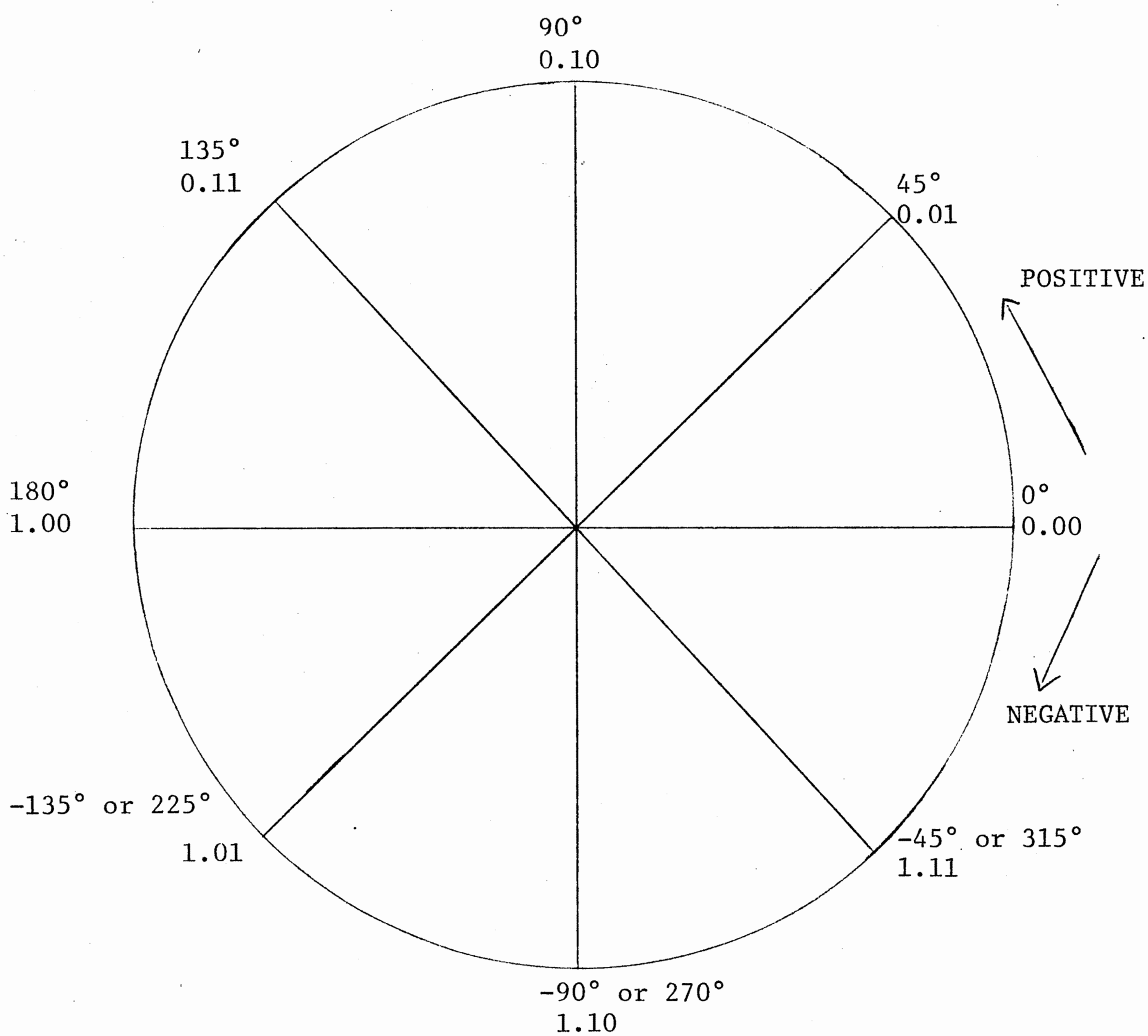

Figure 3. Representation of Angles in Cordic.

CHAPTER IV

COMPUTER IMPLEMENTATION AND PROGRAMMING RESULTS

The four tasks described in Chapter I are performed and the programming results are obtained in this chapter.

The description of the HP21MX computer which is used to aid this research is given below.

System Features

The HP21MX computer is a powerful user-microprogrammable minicomputer with 178 micro-instructions and 4K words of control space. Each word is 24 bits long. It has 128 standard instructions, 80 of which emulate the HP 2100 series computer; 42 of which are new instructions for indexing, byte and bit manipulation, byte and word moves, and byte string scanning; and 6 of which are single-precision floating point instructions. There are four general purpose registers, two of which may be used as index registers. It is a fully microprogrammed processor, including all arithmetic functions, input/output, and operator panel control. Writable Control Store (WCS) is optional.

The read-only memory (ROM) modules in which microprograms are stored are referred to collectively as control store. Standard control consists of 1,024 directly addressable locations configured into four modules of 256 location each. Each control store location accommodates one micro-instruction, which in turn consists of a 24-bit word encompassing six micro-orders. The control store address space of each processor is 4,096 words.

Microprograms in standard control store for executing the various machine functions are divided into three groups:

Base instruction set (modules 0 and 1)

Floating point instructions (module 14)

Extended instruction group (module 15)

Unused modules of control store are available for user-supplied microprograms. Microinstructions in control store are 24 bits long; whereas, machine language instructions residing in main memory are 16 bits long. In addition, microinstructions have access to many internal registers and logic functions that machine language instructions cannot use.

The Writable Control Store (WCS) option provides a read-write control store module which can be used for the development and execution of user-supplied microprograms. Microprograms in WCS are executed at the same speed as those in the read-only control store.

Hardware Registers

A 16-bit accumulator which holds the results of arithmetic and logical operations performed by programmed instructions.

B-register

Serves the same purpose as the A-register, but is independent from it.

M-register

A 16-bit register used to hold the memory address which is currently being accessed by the CPU.

T-register

A 16-bit register used to hold the data which are stored into or retrieved from memory.

P-register

Program counter, 16 bits long, pointing to next instruction to be fetched from memory.

S-register

A 16-bit utility register. In the halt or run mode, it can be loaded via the display register.

Extend register

A one-bit register used to link the A- and B-registers by rotation instructions or to indicate a carry from the most significant bit (bit 15) of the A- or B-register by an add instruction or increment instruction.

Overflow Register

A one-bit register used to indicate that an add instruction, divide instruction, or an increment instruction has caused the A-register or B-register to exceed the maximum positive or negative number that can be contained in these registers.

Display register

A 16-bit register included in the front panel and used to display and modify the contents of the six 16-bit working registers when the computer is in the halt mode.

X- and Y-registers

Two 16-bit registers serving as indexing registers which are accessed through the use of 30 index register instructions and 2 jump instructions.

S_1 to S_{12} scratch pad registers

Twelve registers (each 16 bits long) used to temporarily store data by a microprogram and cannot be accessed by a macro-program.*

Interrupt System

The vectored priority interrupt system has up to 60 distinct interrupt levels, each of which has a unique priority assignment. Each interrupt level is associated with a numerically corresponding interrupt location in memory.

Of the 60 interrupt levels, the first two are reserved for hardware faults (power failure and parity error); the next two are reserved for the Dual-Channel port controller completion interrupts; and the reamining levels are available for I/O device channels. Table III lists the interrupt levels in priority order for the HP 2108 processor of the 21 MX.

APL Description of HP21MX

In the APL description of the HP21MX, the computer system is described as seen by a programmer, and the description is independent of any particular hardware implementation. All those instructions which are not connected with this research are not included in this description. Iverson (2) gives a complete definition of the notation used. The description is based on the HP21MX Computer Series Reference Manual (5) and consists of a set of programs and tables.

---

* Macroprogram - programs stored in main memory.

Microprogram - programs stored in control store.

TABLE III

INTERRUPT ASSIGNMENTS

Channel	Interrupt Location	Assignments
(Octal)		
04	00004	Power Fail Interrupt
05	00005	Memory Parity/Protect Interrupt
06	00006	DCPC Channel 1 Completion Interrupt
07	00007	DCPC Channel 2 Completion Interrupt
10	00010	1/0 Device (highest priority)
11-20	00011-00020	1/0 Device (Mainframe)
21-42	00021-00042	1/0 Device (Extender No. 1)
43-64	00043-00064	1/0 Device (Extender No. 2)

The programs are either system programs or defined operations. All programs operate concurrently and continuously, with one line active in each program. The defined operation program operates only when invoked by another program. In the description presented, PROC and IOIG are system programs, whereas ADC, EXEC, and MAC are defined operations.

The Processor

The PROC program, Figure 4, describes the sequencing and execution of instructions and the servicing of interrupts. The program segments and their functions are summarized in Table IV.

TABLE IV

"PROC" PROGRAM SEGMENTS

Lines	Function
1-4	Instruction fetch
5-14	Instruction decoding
15-26	Instruction execution
27-30	Trap interrupt service

Instruction Fetch

The first step in program execution is to fetch the instruction from memory. In order to prepare for instruction fetch, the exceptions vector is initialized to zero (line 1). The 16-bit instruction is fetched from memory at the address given by the program counter, and placed in the instruction register (line 2). The program counter is incremented by 2 (line 3), and in case of any exceptions during instruction fetch, control branches to line 27. Exceptions during fetch may be due to errors in parity check.

PROC system program

$1 : i\ \rho 2$ 0

$S \leftarrow 64$ 0.1

$e \leftarrow (4)$ 1

$MAC^1(\iota P,\ f;\ T)$ 2

$P \leftarrow (16)\top 1 + P$ 3

$1 : \vee/e$ 4

$i \leftarrow 1$ 5

$j \leftarrow 0$ 6

$\phi : \iota\ Ui \oplus (Qi \wedge I)$ 7

$i \leftarrow i + 1$ 8

$12 : i$ 9

$j \leftarrow i$ 10

$k_1 \leftarrow \iota\ (\epsilon^{1,2,3}/I,\ \epsilon^{15}/I,\ \epsilon^{12}/I,\ \epsilon^{15}/I,\ \epsilon^{14,15}/I,\ \epsilon^{13,14,15}/I\ \epsilon^{11,13,14,15}/I,\ \epsilon^{10,11}/I,\ \epsilon^{9,10,11}/I,\ \epsilon^{4}/I,\ \epsilon^{7,8,9}/I,\ \epsilon^{6,7}/I,\ \epsilon^{7,8,9}/I)_j$ 11

$k_2 \leftarrow \iota\ (\epsilon^{4}/I,\ 0,\ 0,\ \epsilon^{14}/I,\ \epsilon^{13}/I,\ \epsilon^{4,11,12}/I,\ \epsilon^{4,12}/I,\ \epsilon^{4}/I,\ \epsilon^{6}/I,\ \epsilon^{7}/I,\ \epsilon^{4}/I,\ \epsilon^{4}/I,\ \epsilon^{4}/I)_j$ 12

$m \leftarrow {}^{j}D^{k_2}_{k_1}$ 13

$n \leftarrow N^m$ 14

$\rightarrow (16,\ 25,\ 25,\ 25,\ 25,\ 25,\ 25,\ 25,\ 25,\ 20,\ 21,\ 22,\ 22)_j$ 15

$t \leftarrow \lfloor((\iota P)-1) \div 1024$ 16

$b \leftarrow (\iota\omega^{10}/I) + I_5 \times t \times 1024$ 17

$ADC\ (I_0 : b;\ a)$ 18

$1 : \vee/e$ 19

$a \leftarrow \iota\omega^4/I$ 20

$(N^m)_1 \leftarrow I_6$ 21

$K_1 \leftarrow (\ ^{13,14,15}/I,\ \ ^{8,9}/I)_{I5}$ 22

$j \leftarrow j + (0,\ 2)_{I5}$ 23

$m \leftarrow {}^{j}D^{k_2}_{k_1}$ 24

$v \leftarrow N^m$ 25

EXEC 26

$0 : \vee/e$ 27

$h_0 \leftarrow 1$ 28

$0 : \vee/h$ 29

$MAC^1((4,5,6,7,\iota\omega^6/T)_{((e,h_1)/\iota\ ^0)_0},f:I)$ 30

$h_{(h/\iota\ ^0)_0} \leftarrow 0$ 31

$e \leftarrow \bar{\epsilon}(4)$

Figure 4. The Processor System Program

Instruction Decoding

To determine the operation specified by the instruction, the instruction is decoded next. Because the operation code of an instruction in this machine may be varied from 4 bits to 16 bits and several microinstructions may be involved in a single instruction word for some type of instructions, the decoding task is very complicated and tedious. Many steps and two sets of decoding vectors named u and q are used in this APL description to aid the decoding task. These two sets of vectors ar listed in Table V. The instructions are divided into 13 classes. Table IV summarizes those 13 classes. The number involved in this table is used to identify the class of the instruction during the decoding.

The class identifiers j and i are initialized in line 5 and 6. The decoding vectors U_i and E_i are used in lines 7, 8, and 9 to identify the class of the current instruction. Once the class of the current instruction is found, it is stated in j (line 10).

The components of the selection vector k take on the values of the fields depending on j (lines 11 and 12). Lines 13 and 14 interpret the instruction by selecting a row N^i from the navigation matrix N (Table VII), to specify the vector n used in subsequent control of the instruction execution. The row of N selected, is determined by an element of a particular decoding matrix D, Figure 6, specified by the instruction class j, and the selection vector k.

TABLE V

DECODING VECTORS

WMI	U_1 = (1000101111111110)	Q_1 = (1111111111111110)
JMPI	U_2 = (1000101111110010)	Q_2 = (1111111111110111)
BIMI	U_3 = (1000101111111000)	Q_3 = (1111111111111000)
BYMI	U_4 = (1000101111110000)	Q_4 = (1111111111111000)
DMI	U_5 = (1000001111000000)	Q_5 = (1111011111100000)
IRI	U_6 = (1000001111100000)	Q_6 = (1111011111100000)
FRI	U_7 = (1000101000000000)	Q_7 = (1111010000000000)
EAMR	U_8 = (1000000000000000)	Q_8 = (1111010001110000)
EAR	U_9 = (1000000000000000)	Q_9 = (1111010000000000)
IOI	U_{10} = (0000010000000000)	Q_{10} = (1111010000000000)
A/S	U_{11} = (0000010000000000)	Q_{11} = (1111010000000000)
S/R	U_{12} = (0000000000000000)	Q_{12} = (1111010000000000)

TABLE VI

INSTURCTION CLASSES

Class		j
MRI:	Memory reference instructions	0
WMI:	Word manipulation instructions	1
MJPI:	Jump instructions	2
BIMI:	Bit manipulation instructions	3
BYMI:	Byte manipulation instructions	4
DMI:	Dynamic mapping system instructions	5
IRI:	Index register instructions	6
FPI:	Floating point instructions	7
EAMR:	Extended arithmetic memory reference instructions	8
EAR:	Extended arithmetic register reference instructions	9
IOI:	Input/output instructions	10
A/S:	Alter skip instructions	11
S/R:	Shift/rotate instructions	12

TABLE VII

THE NAVIGATION MATRIX

n_0	n_1	n_2	n_3	Class	Index	Mnemonic	Name	Op Code
0	0	a_0	a_3	MRI	1	ADA	Add to A	-1000 -----------
1	0	a_0	a_3	MRI	2	ADB	Add to B	-1001 -----------
0	-	b_0	b_5	IRI	3	ADX	Add memory to X	1000101111100110
1	-	b_0	b_5	IRI	4	ADY	Add memory to Y	1000101111101110
0	-	e_0	e_3	S/R	5	ALF	Rotate A left four	0000001111-1-111
0	-	e_0	e_4	S/R	6	ALR	A left shift. clear sign	0000001100-1-100
0	-	e_0	e_5	S/R	7	ALS	A left shift	0000001000-1-000
0	0	a_0	a_2	MRI	8	AND	"AND" to A	-0010 ----------
0	-	e_0	e_6	S/R	9	ARS	A right shift	0000001001-1-001
-	-	c_0	-	EAR	10	ASL	Arithmetic shift left	100000000001 ---
-	-	c_1	-	EAR	11	ASR	Arithmetic shift right	100000100001 ---
1	-	e_0	e_3	S/R	12	BLF	Rotate B left four	0000101111-1-111
1	-	e_0	e_4	S/R	13	BLR	B left shift, clear sign	0000101100-1-100

TABLE VII (Continued)

n_0	n_1	n_2	n_3	Class	Index	Mnemonic	Name	Op Code
1	-	e_0	e_5	S/R	14	BLS	B left shift	0000101000-1-000
1	-	e_0	e_1	S/R	15	BRS	Bright shift	0000101001-1-001
0	2	b_4	b_8	IRI	16	CAX	Copy A to X	1000001111100001
0	3	b_4	b_8	IRI	17	CAY	Copy A to Y	1000001111111100
-	-	-	-	BIMI	18	CBS	Clear bits	1000101111111100
-	-	-	-	BYMI	19	CBT	Compare bytes	1000101111110110
1	2	b_4	b_8	IRI	20	CBX	Copy B to X	1000101111101001
1	3	b_4	b_8	IRI	21	CBY	Copy B to Y	1000101111101001
0	-	f_0	-	A/S	22	CCA	Clear and complement A	00000111 -------
1	-	f_0	-	A/S	23	CCB	Clear and complement B	00001111 -------
-	-	f_7	-	A/S	24	CCE	Clear and complement E	0000-1--11 -----
0	-	f_2	-	A/S	25	CLA	Clear A	00000101 -------
1	-	f_2	-	A/S	26	CLB	Clear B	
-	0	d_0	d_2	IOI	27	CLC	Clear control	100011-111 -----
-	-	f_5	-	A/S	28	CLE	Clear E	0000-1--01 -----
-	0	d_0	d_6	IOI	29	CLF	Clear flag	1000-11001 -----
-	-	-	-	IOI	30	CLO	Clear overflow	1000011001000001

TABLE VII (Continued)

n_0	n_1	n_2	n_3	Class	Index	Mnemonic	Name	Op Code
1	–	B_4	B_9	IRI	47	DSY	Decrement Y and skip if zero	1000101111111001
0	0	e_0	e_7	S/R	48	ELA	Rotate E left with A	0000001110-1-110
1	0	e_0	e_7	S/R	49	ELB	Rotate E left with B	0000101110-1-110
0	1	e_0	e_7	S/R	50	ERA	Rotate E right with A	0000001101-1-101
1	0	e_0	e_7	S/R	51	ERB	Rotate E right with B	0000101101-1-1-1
–	–	–	–	FPI	52	FAD	Floating point add	1000101000000000
–	–	–	–	FPI	53	FDV	Floating point divide	1000101000110000
–	–	–	–	FPI	54	FIX	Floating point to integer	1000101001000000
–	–	–	–	FPI	55	FLT	Integer to floating point	1000101001010000
–	–	–	–	FPI	56	FMP	Floating point multiply	1000101000100000
–	–	–	–	FPI	57	FSB	Floating point subtract	1000101000010000
–	0	d_0	d_{11}	IOI	58	HLT	Halt	1000-1-000 -----
–	–	–	–	A/S	59	INA	Increment A	000001-------1--
–	–	–	–	A/S	60	INR	Increment B	000011-------1--
1	0	a_0	a_2	MRI	61	IOR	"Inclusive OR" to A	-011------------
0	0	b_4	b_9	IRI	62	ISX	Increment X and skip if zero	1000101111110000
0	1	b_4	b_9	IRI	63	ISY	Increment Y and skip if zero	1000101111111000

TABLE VII (Continued)

n_0	n_1	n_2	n_3	Class	Index	Mnemonic	Name	Op Code
-	-	-	-	A/S	31	CMA	Complement A	00000110 --------
-	-	-	-	A/S	32	CMB	Complement B	00001110 --------
-	-	f6	-	A/S	33	CME	Compare E	0000-1--10 -----
-	-	-	-	WMI	34	CMW	Compare words	1000101111111110
0	-	a_0	a_8	MRI	35	CPA	Compare to A	-1010 -----------
1	-	a_0	a_8	MRI	36	CPB	Compare to B	-1011 -----------
2	0	b_4	b_8	IRI	37	CXA	Copy X to A	1000001111100100
2	1	b_4	b8	IRI	38	CXB	Copy X to B	1000101111100100
3	0	b_4	b_8	IRI	39	CYA	Copy Y to A	1000001111101100
3	1	b_4	b_8	IRI	40	CYB	Copy Y to B	1000101111101100
-	-	-	-	EAMR	41	DIV	Divide	100000010000 ---
-	-	-	-	DMI	42	DJP	Disable mem and jump	1000101111011010
-	-	-	-	DMI	43	DJS	Disable mem and jump to sub- routine	1000101111011011
-	-	-	-	EAMR	44	DLD	Double load	100010001000 ---
-	-	-	-	EAMR	45	DST	Double store	100010010000 ---
1	0	b_4	b_9	IRI	46	DSX	Decrement X and skip if zero	100010111110001

TABLE VII (Continued)

n_0	n_1	n_2	n_3	Class	Index	Mnemonic	Name	Op Code
–	–	a_0	a_9	MPI	64	ISZ	Increment and skip if zero	-0111-----------
–	–	g_0	–	JMPI	65	JLY	Jump and load Y	1000101111110010
1	–	a_1	a_{13}	MPI	66	JMP	Jump	-0101-----------
–	–	g_4	–	JMPI	67	JPY	Jump indexed by Y	1000101111111010
–	–	–	–	DMI	68	JRS	Jump and store status	1000101111001101
0	–	a_1	a_{12}	MPI	69	JSB	Jump to subroutine	-0011-----------
0	0	b_0	b_{11}	IRI	70	LAX	Load A indexed by X	1000001111100010
0	1	b_0	b_{11}	IRI	71	LAY	Load A indexed by Y	1000001111101010
–	–	–	–	BYMI	72	LBT	Load byte	1000101111110011
1	0	b_0	b_{11}	IRI	73	LBX	Load B indexed by X	1000101111000010
1	1	b_0	b_{11}	IRI	74	LBY	Load B indexed by Y	1000101111101010
0	0	a_1	a_7	MRI	75	LDA	Load A	-1100-----------
1	0	a_1	a_7	MRI	76	LDB	Load B	-1101-----------
0	–	b_0	b_{12}	IRI	77	LDX	Load X from memory	1000101111100101
1	–	b_0	b_{12}	IRI	78	LDY	Load Y from memory	1000101111101101
–	–	–	–	DMI	79	LFA	*Load fence from A	1000001111010111
–	–	–	–	DMI	80	LFB	*Load fence from B	1000101111010111

TABLE VI (Continued)

n_0	n_1	n_2	n_3	Class	Index	Mnemonic	Name	Op Code
0	0	d_0	d_{12}	IOI	81	LIA	Load into A	100001-101------
1	0	d_0	d_{12}	IOI	82	LIB	Load into B	100011-101------
-	-	c_2	-	EAR	83	LSL	Logical shift left	10000000001-----
-	-	c_3	-	EAR	84	LSR	Logical shift right	100000100010----
-	-	-	-	DMI	85	MBF	Move bytes from alternate map	1000101111000011
-	-	-	-	DMI	86	MBI	Move bytes into alternate	1000101111000010
-	-	-	-	BMI	87	MBT	Move bytes	1000101111110101
-	-	-	-	DMI	88	MBW	Move bytes within alternate	1000101111000100
0	0	d_0	d_{13}	IOI	89	MIA	Merge into A	100001-100------
1	-	d_0	d_{13}	IOI	90	MIB	Merge into B	100011-100------
-	-	-	-	EAMR	91	MPY	Multiply	100000001000----
-	-	-	-	WMI	92	MVW	Move words	1000101111111111
-	-	-	-	DMI	93	MWF	Move words from alternate map	1000101111000110
-	-	-	-	DMI	94	MWI	Move words into alternate map	1000101111000101
-	-	-	-	DMI	95	MWW	Move words within alternate map	1000101111000111
-	-	-	-	S/R	96	NOP	No Operation	0000000000000000

TABLE VII (Continued)

n_0	n_1	n_2	n_3	Class	Index	Mnemonic	Name	Op Code
0	-	d_0	d_{20}	IOI	97	OTA	Output A	100001-110------
-	0	d_0	d_{20}	IOI	98	OTB	Output B	100011-110------
-	-	-	-	DMI	99	PAA	Load/store port A map per A	1000001111001010
-	-	-	-	DMI	100	PAB	Load/store port A map per B	1000101111001010
-	-	-	-	DMI	101	PBA	Load/store port B map per A	1000001111001011
-	-	-	-	DMI	102	PBB	Load/store port B map per B	10001-1111001011
0	0	e_0	e_9	S/R	103	RAL	Rotate A left	0000001010-1-010
1	0	e_0	e_9	S/R	104	RAR	Rotate A right	0000001011-1-011
0	1	e_0	e_9	S/R	105	RBL	Rotate B left	0000101010010010
1	1	e_0	e_9	S/R	106	RBR	Rotate B right	0000101011-1-011
-	-	c_4	-	EAR	107	RRL	Rotate left	100000000100----
-	-	c_5	-	EAR	108	RRR	Rotate right	100000100100----
-	-	-	-	DMZ	109	RSA	Read status register into A	1000001111011000
-	-	-	-	DMI	110	RSB	Read status register into B	1000101111011000
-	-	-	-	A/S	111	RSS	Reverse skip sense	0000-1---------1
-	-	-	-	DMI	112	RVA	Real violation register into A	1000001111011001

TABLE VII (Continued)

n_0	n_1	n_2	n_3	Class	Index	Mnemonic	Name	Op Code
–	–	–	–	DMI	113	RVB	Read violation register into B	1000101111011001
0	0	b_0	b_{14}	IRI	114	SAX	Store A indexed by X	1000001111100000
0	0	b_0	b_{14}	IRI	115	SAY	Store A indexed by Y	1000001111101000
–	–	–	–	BIMI	116	SBS	Set bits	1000101111111011
–	–	–	–	BYMI	117	SBT	Store type	1000101111110100
1	0	b_0	b_{14}	IRI	118	SBX	Store B indexed by X	1000101111100000
1	1	b_0	b_{14}	IRI	119	SBY	Store B indexed by Y	1000101111101000
–	–	–	–	A/S	120	SEZ	Skip if E is zero	0000-1----1-----
–	–	–	–	BYMI	121	SFB	Skip if flag clear	1000-10010------
–	0	d_0	d_{14}	IOI	122	SFC	Skip if flag clear	1000-10011------
–	0	d_0	d_{16}	IOI	123	SFS	Skip if flag set	1000-10011------
–	–	–	–	DMI	124	SJP	Enable system map and jump	1000101000100000
–	–	–	–	DMI	125	SJS	Enable system map and jump to subroutine	1000101111011101
–	–	–	–	S/R	126	SLA	Skip if LSB of A is zero	00000-------1---
–	–	–	–	S/R	127	SLB	Skip if LSB of B is zero	000010------1---
–	0	d_0	d_{14}	IOI	128	SOC	Skip if overflow clear	100001-010000001

TABLE VII (Continued)

n_0	n_1	n_2	n_3	Class	Index	Mnemonic	Name	Op Code
-	0	d_0	d_{16}	IOI	129	SOS	Skip if overflow set	100001-011000001
-	-	-	-	A/S	130	SSA	Skip if sign of A is zero	000001-----1----
-	-	-	-	A/S	131	SSB	Skip if sign of B is zero	000011-----1----
-	-	-	-	DMI	132	SSM	Store status register into memory	1000101111001100
0	1	a_1	a_7	MRI	133	STA	Store A	-1110-----------
1	1	a_1	a_7	MRI	134	STB	Store	-1111-----------
-	0	d_0	d_{18}	IOI	135	STC	Set control	100001-111------
-	0	d_0	d_{19}	IOI	136	STF	Set flag	1000-10001------
-	0	d_0	d_{19}	IOI	137	STO	Set overflow	1000010001000001
0	-	b_0	b_{13}	IRI	138	STX	Store X to memory	1000101111100011
1	-	b_0	b_{13}	IRI	139	STY	Store Y to memory	1000101111101011
-	-	-	-	DMI	140	SYA	Load/store system map per A	1000001111001000
-	-	-	-	DMI	141	SYB	Load/store system map per B	1000101111001000
-	-	-	-	A/S	142	SZA	Skip if A is zero	000001--------1-
-	-	-	-	A/S	143	SZB	Skip if B is zero	000011--------1-
-	-	-	-	BYMI	144	TBS	Test bits	1000101111111101

TABLE VII (Continued)

n_0	n_1	n_2	n_3	Class	Index	Mnemonic	Name	Op Code
–	–	–	–	DMI	145	UJP	Enable user map and jump to subroutine	1000101111011110
–	–	–	–	DMI	146	UJS	Enable user map and jump to subroutine	1000101111011111
–	–	–	–	DMI	147	USA	Load/store user map per A	1000001111001001
–	–	–	–	DMI	148	USB	Load/store user map per B	1000101111001001
0	0	b_4	b_{15}	DMI	149	XAX	Exchange A and X	1000001111100111
0	1	b_4	b_{15}	IRI	150	XAY	Exchange A and X	1000001111101111
1	–	b_4	b_{15}	IRI	151	XBX	Exchange B and X	1000101111100111
1	1	b_4	b_{15}	IRI	152	XBY	Exchange B and Y	1000101111101111
–	–	–	–	DMI	153	XCA	Cross compare A	1000001111010110
–	–	–	–	DMI	154	XCB	Cross Compare B	1000101111010110
–	–	–	–	DMI	155	XLA	Cross load A	1000001111010100
–	–	–	–	DMI	156	XLB	Cross load B	1000101111010100
–	–	–	–	DMI	157	XMA	Transfer maps internally per A	1000101111010000
–	–	–	–	DMI	158	XMB	Transfer maps internally per B	1000101111010010
–	–	–	–	DMI	159	XMM	Transfer maps or memory	1000101111010000

TABLE VII (Continued)

n_0	n_1	n_2	n_3	Class	Index	Mnemonic	Name	Op Code
-	-	-	-	DMI	160	XMS	Transfer maps sequentially	1000101111010001
-	-	-	-	MPI	161	XOR	"Exclusive OR" to A	-0100-----------
-	-	-	-	DMI	162	XSA	Cross Store A	1000001111010101
-	-	-	-	DMI	163	XSB	Cross store B	1000101111010101
0	2	b_4	b_8	IRI	164	CAX	Copy A to X	1000001111100001
-	3	b_4	b_8	IRI	165	CAY	Copy A to Y	1000001111101001
1	2	b_4	b_8	IRI	166	CBX	Copy B to X	1000101111100001
1	3	b_4	b_8	IRI	167	CBY	Copy B to Y	1000101111101001

* Base page fence register

$1 \; : \; \vee/F$	0
$h_1 \leftarrow (S > (F/\iota^0)_0)$	1
$1 \; : \; h_1$	2
$\omega 6/T \leftarrow \top(F/\iota^0)_0$	3
$S \leftarrow (F/\iota^0)_0$	4

Figure 5. Input/Output Interrupt Generator

Instruction Execution

The instruction execution starts at line 15. The effective address computation of MRI is performed at lines 16, 17, 18 and 19. Line 20 sets up the immediate value for EAR. Line 21 sets up I/O flag clear/hold information for IOI. Line 22-24 subdecodes the packed micro-instructions in A/S and S/R instructions.

Interrupt Service

Servicing of exceptions is given priority over I/O interrupt service. In case of any exception the bit (0 for exception, 1 for I/O interrupt) in the interrupt holder h is set (line 27). The interrupt service sequence is initiated if at least one interrupt is pending (line 28). The sequence consists of fetching a new instruction from one of the five fixed locations in memory. The interrupt vector address of the peripheral device is obtained from the six least significant bits of the T-bus. The element of h which caused the interrupt is reset.

	0	1
0	CMW 34	MVW 92

1_D

(a) WMI Instruction

	0	1	2	3	4	5	6	7
0								
1	140 SYA	147 USA	99 PAA	101 PBA				
2			157 XMA	155	153 XLA	79 XSA	XCA	LFA
3	109 RSA	112 RVA						
4			86 MBI	85 MBF	88 MBW	94 MWI	93 MWF	95 MWW
5	141 SYB	148 USB	100 PAB	102 PBB	132 SSM	68 JRS		
6	159 XMM	160 XMS		158 XMB	156 XLB	163 XSB	154 XCB	80 LFB
7	110 RSB	113 RVB	42 DJP	43 DJS	124 SJP	125 SJS	145 UJP	146 UJS

5_D

(b) DMSI Instruction

	0	1	2	3	4	5	6	7	8	9	10	 15
0	114 SAX	16 CAX	70 LAX		37 CXA			149 XAX				
1	115 SAY	17 CAY	71 LAY		39 CYA			150 XAY				
2	118 SBX	20 CBX	73 LBX	138 STX	38 CXB	77 LDX	3 ADX	151 XBX	62 ISX	46 DSX		
3	119 SBY	21 CBY	74 LBY	139 STY	40 CYB	78 LDY	4 ADY	152 XBY	63 ISY	47 DSY		

6_D

(c) IRI Instruction

Figure 6. Instruction Decoding Matrices

		0	1	2	3	4	5	6	7
I_4	0		8 AND	161 XOR	61 IOR	1 ADA	35 CPA	75 LDA	133 STA
	1		69 JSB	66 JMP	64 ISZ	2 ADB	36 CPB	76 LDB	134 STB

0_D

(d) MRI Instruction

	0	1
0	JLY	JPY

2_D

(e) JMPI Instruction

		0	1	2	3
I_{13}	0				72 LBT
	1	117 SBT	87 MBT	14 CBT	121 SFB

4_D

(f) BYMI Instruction

		0	1
I_{14}	0	18 CBS	144 TBS
	1		116 SBS

3_D

(g) BIMI Instruction

	0	1	2	3
0	52 FAD	57 FSB	56 FMP	53 FDV
1	54 FIX	55 FLT		

7_D

(h) FPI Instruction

Figure 6. (Continued)

	0							7
0		10 ASL	83 LSL		103 RRL			
1		11 ASR	84 LSR		104 RRR			

8_D

(i) EAR Instruction

0	91 MPY	44 DLD
1	41 DIV	45 DST

9_D

(j) EAMR Instruction

	0	1	2	3	4	5	6	7
0	58 HLT	136 STF	122 SFC	123 SFS	89 MIA	81 LIA	97 OTA	27 CLC
1	58 HLT	29 CLF	128 SOC	129 SOS	90 MIB	82 LIB	98 OTB	135 STC

10_D

(k) IOI Instruction

	0	1	2	3
0		25 CLA	31 CMA	22 CCA
1		26 CLB	32 CMB	23 CCB

11_D

(ℓ) A/S Instruction

0	7 ALS	9 ARS	103 RAL	104 RAR	6 ALR	50 ERA	48 ELA	5 ALF
1	14 BLS	15 BRS	105 RBL	106 RBR	13 BLR	51 ERB	49 ELB	12 BLF

12_D

(m) S/R Instruction

Figure 6. (Continued)

Input/output Interrupts

The I/O interrupt generator (IOIG) system program, Figure 5, determine the presence of interrupt requests by peripheral devices and sets the bit in the interrupt holder, h, accordingly (line 1). The dwell at line 0 checks for interrupts on the device flag. The setting of any I/O device flag means an interrupt request from that I/O device. If a higher priority device has already gained control of the processor, the lower priority device cannot be served until the higher priority device has finished its service routine (lines 1, 2, and 3).

Memory Access Routine

The memory access (MAC) operation, Figure 7, fetches or stores a specified number of bytes from the memory at a given address. The general form of the operation is MAC^i (j;1), where i specifies the device requesting access; j is a two-component vector specifying the address in memory (j_o) and the type of operation (store; j_1 = 2; fetch:j_1 = f), respectively; and 1 specified the vector into/from which the accessed data are to be stored/fetched.

The request for service is entered in the bus request vector r, and in the queue if it is empty (line 0). The program dwells at line 1 until i is recognized as the first nonzero entry in the queue. After the request has been honored, the entry in the request vector is blanked out (line 2). The parity error exception is noted (line 5), and entered in the exception vector e. If no exception occurs, a fetch (line 4) or store (line 7) is performed.

Address Computation Routine

The address computation (ADC) operation, Figure 8, is used for effective address calculation of the operands. The general form for ADC is (m; g; k) where m is the mode of addressing (0 means direct, 1 means indirect), g is the primary address, and k is the effective address returned by the operation.

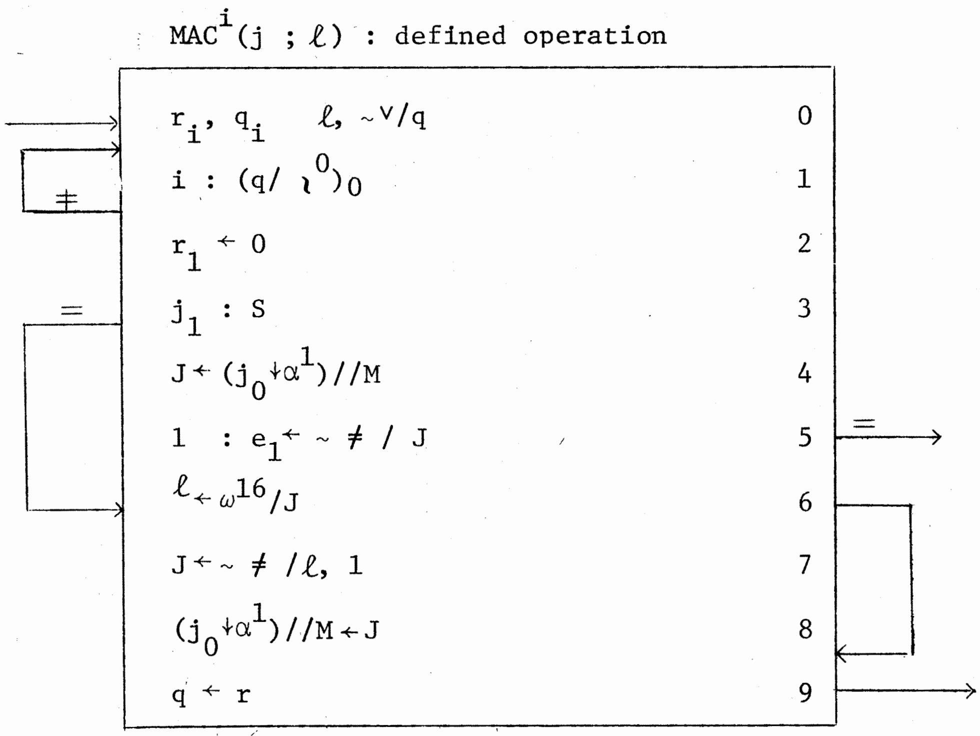

Figure 7. Memory Access Operation

ADC(m ; d ; k) : defined operation

0 : m	0
$k \leftarrow d_1$	1
$MAC^1(d_1, f; \ell)$	2
$k \leftarrow \perp 1$	3

Figure 8. Address Computation Operation

Instruction Execution Routine

At the entry point EXEC, Figure 9, the routine for an instruction is determined by n_2 (line a_0). Execution involves setting up condition codes (if necessary) after the execution.

Lines a_1 - a_{13}

All MRI instructions are executed here. AND, IOR, XOR ADA, ADB, CPA, CPB, and ISZ are entered at line a_1 to get data from memory. STA, STB, LDA, and LDB are entered at line a_2. All MRI instructions are diverged at line a_2 and enter their own routine. The "Exit" here means go back to PROC ; the outgoing arrow at the right side of the line also indicates return to PROC if the arrow does not direct to any other line. This is true not only here, but also in any other line of the EXEC routine.

	$\longrightarrow n_2$	a_0	
	$MAC^1(a,f;C)$	a_1	
	$\longrightarrow n_3$	a_2	
AND, IOR, XOR $\longrightarrow$	$A \longleftarrow A(\wedge,\vee,\oplus)_{n_0}\ C$	a_3	$\longrightarrow$
ADA, ADB $\longrightarrow$		a_4	
	$E,U \longleftarrow ((17)\top(\bot(A,B)_{n_0}) + \bot C)$	a_5	
	$0 \longleftarrow (u_a \oplus C_0)\wedge(\sim((A,B)_{n_0} \oplus C_0)$	a_6	
STA, STB $\longrightarrow$	$(A,B)_{n_0} \longleftarrow u$	a_7	$\longrightarrow$
LDA.LDB	$MAC^1(a,(f,s)_{n_1};\ (A,B)_{n_0})$	a_8	$\longrightarrow$
CPA $\longrightarrow$ CPB	$\longrightarrow (a_{11}, \text{Exit})_{(0 = \bot\ (A,B)_{n_1} \oplus C)}$	a_9	
ISZ $\longrightarrow$	$C \longleftarrow \top 1 + \bot C$	a_{10}	
	$MAC^1(a,s;C)$		
	$0 : \bot C$	a_{11}	$\xrightarrow{=}$
	$P \longleftarrow \top 1 + \bot P$	a_{12}	$\longrightarrow$
JSB $\longrightarrow$	$MAC^1(a,s;P)$	a_{13}	
JMP $\longrightarrow$	$P \longleftarrow \top\ a + (1,0)_{n_0}$	a_{14}	$\longrightarrow$
	$MAC^1(\bot P,\ f\ ;\ C)$	b_0	
	$P \longleftarrow \top 1 + \bot P$	b_1	
	$ADC\ (C_0\ ;\bot\omega\ ^{15}/C\ ;\ a)$	b_2	
	$MAC^1(a,\ f\ ;\ C)$	b_3	
	$\longrightarrow n_3$	b_4	
ADX, ADY	$E,u \longleftarrow (17)\top(\bot(X,Y)_{n_0} + \bot\ C)$	b_5	
	$0 \longleftarrow (u_0 \oplus C_0)\ (\wedge(\sim(X,Y)_{n_0})_0 \oplus C_0)$	b_6	

Figure 9. EXEC Routine.

CXA,CXB	$(X,Y)_{n_0} \leftarrow u$	b_7	=
CBX,CAX CBY,CAY CYA,CYB	$(A,B,X,Y)_{n_1} \leftarrow (A,B,X,Y)_{n_0}$	b_8	
ISX,DSX ISY,DSY	$0 : (X,Y)_{n_1} \leftarrow \top(1,-1)_{n_0} + \perp(X,Y)_{n_1}$	b_9	≠
	$P \leftarrow \top 1 + \perp P$	b_{10}	
LAX,LAY LBX,LBY	$MAC^1(a + (X,Y)n_1,\ f;\ (A,B)_{n_0})$	b_{11}	
LDX,LDY	$(X,Y)_{n_0} \leftarrow C$	b_{12}	
STX,STY	$MAC^1\ (a,s,(X,Y)n_0)$	b_{13}	
SAX,SAY SBX,SBY	$MAC^1\ (b + \perp(X,Y)_{n_1},\ s;\ (A,B)_{n_0})$	b_{14}	
XAX,XAY XBX,XBY	$C \leftarrow (A,B)_{n_0}$	b_{15}	
	$(A,B)_{n_0} \leftarrow (X,Y)_{n_1}$	b_{16}	
	$(X,Y)_{n_1} \leftarrow C$	b_{17}	
ASL	$\omega^{31}/(B,A) \leftarrow a \overset{\uparrow}{\circ} \omega^{31}/(B,A)$	c_0	
ASR	$\omega^{31}/(B,A) \leftarrow (\epsilon(31),\alpha^a(31))_{B_0}$ $\vee\ (a \underset{\downarrow}{\circ} \omega^{31}/(B,A))$	c_1	
LSL	$B,\ A \leftarrow a \overset{\uparrow}{\circ} (B,\ A)$	c_2	
LSR	$B,\ A \leftarrow a \underset{\downarrow}{\circ} (B,\ A)$	c_3	
RRL	$B,\ A \leftarrow a \uparrow (B,\ A)$	c_4	
RRR	$B,\ A \leftarrow a \downarrow (B,\ A)$	c_5	

Figure 9. (Continued)

	$a \leftarrow \omega^{6}/I$	d_0	
	$\longrightarrow n_3$	d_1	
CLC $\longrightarrow$	$V_a \leftarrow \phi$	d_2	
	$S \leftarrow 63$	$d_{2.1}$	
	$0 : a$	d_3	$\xrightarrow{\neq}$
	$V \leftarrow \bar{\varepsilon}(56)$	d_4	
	$0 : n_1$	d_5	$\xrightarrow{=}$
CLF $\longrightarrow$	$1 : a$	d_6	
	$F_a \leftarrow \phi$	d_7	
	$0 : a$	d_8	
	$F \leftarrow \bar{\varepsilon}(58)$	d_9	$\longrightarrow$
$\longrightarrow$	$0 \leftarrow \phi$	d_{10}	
HLT $\longrightarrow$	$RUN \leftarrow \phi$	d_{11}	$\longrightarrow$
LIA,LIB $\longrightarrow$	$(A,B)_{n_0} \leftarrow Z_a$	d_{12}	$\longrightarrow$
MIA,MIB $\longrightarrow$	$(A,B)_{n_0} \leftarrow (A,B)_{n_0} \vee Z_a$	d_{13}	$\longrightarrow$
SFC,SOC $\longrightarrow$	$P \leftarrow \top(0 = (0,F_a)_{(a \neq 0)}) + \bot P$	d_{14}	
	$\longrightarrow (d_5,\ \text{Exit})_a$	d_{15}	
SFS / SOS $\longrightarrow$	$P \leftarrow \top(1 = (0,F_a)_{(a \neq 0)}) + \bot P$	d_{16}	
	$\longrightarrow (d_3,\ \text{Exit})_a$	d_{17}	
STC $\longrightarrow$	$V_a \leftarrow 1$	d_{18}	
STF / STO	$(F_a,\ 0)_{(a \neq o)} \leftarrow 1$	d_{19}	

Figure 9. (Continued)

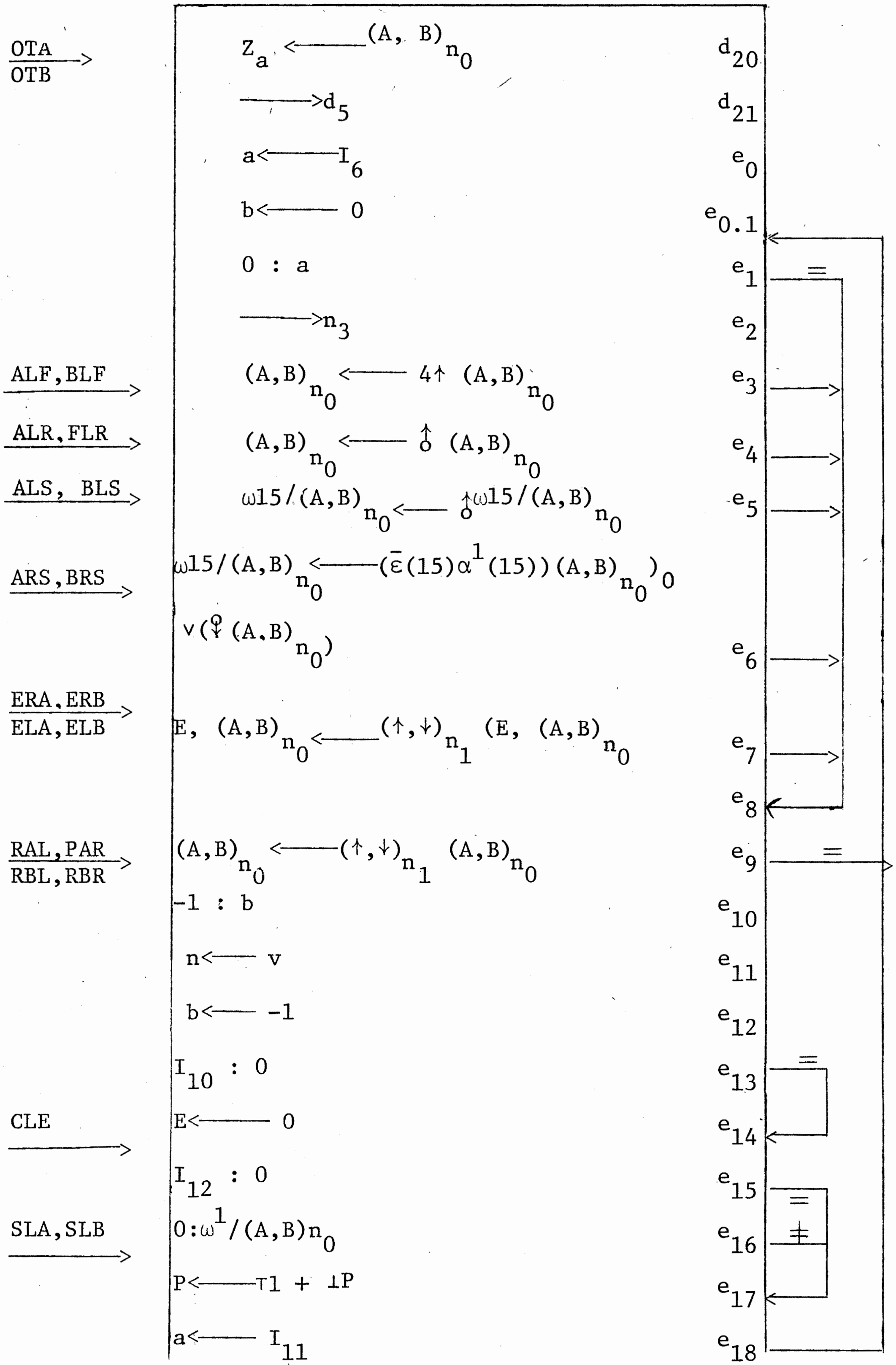

Figure 9. (Continued)

CCA / CCB	$(A,B)_{n_0} \leftarrow \epsilon(16)$	f_0
CMA / CMB	$(A,B)_{n_0} \leftarrow \sim(A,B)_{n_0}$	f_1
CLA / CLB	$(A,B)_{n_0} \leftarrow \bar{\epsilon}\ (16)$	f_2
	$n \leftarrow v$	f_3
	$\rightarrow n_2$	f_4
CLE	$E \leftarrow 0$	f_5
CME	$E \leftarrow \bar{E}$	f_6
CCE	$E \leftarrow 1$	f_7
	$S \leftarrow 0$	f_8
SEZ, SLA SLB, SSA	$S \leftarrow (\bar{E} \wedge I_{10}) \vee ((0=\alpha^1/(A,B)I_4) \wedge I_{11})$ $\vee((0=\omega^1/(A,B)I_4) \wedge I_{12}) \vee ((0=R(A,B)I_4)$ $\wedge I_{14})$	f_9
INA / INB	$(A,B)_{I_4} \leftarrow \top(0,a)_{I_{13}} + \bot(A,B)_{I_4}$	f_{10}
RSS	$S \leftarrow (S,\bar{S})I_{15}$	f_{11}
	$P \leftarrow \top(0,1)_{I_{15}} + \bot P$	f_{12}
JLY	$Y \leftarrow \top 1 + \bot P$	g_0
	$MAC^1(\bot P, f; I)$	$g_\emptyset$
	$\rightarrow n_3$	g_1
JLY ------	$Y \leftarrow \top 1 + \bot P$	g_2
	$ADC(I_\emptyset; \bot\omega^{15}/I; a)$	g_3
	$P \leftarrow \top a$	g_4
JPY ------	$p \leftarrow (\bot Y) + (\bot\omega^{15}/I)$	g_5

Figure 9. (Continued)

	$MAC^1(\perp P, f; I)$	h_0	
	$P \longleftarrow \top 1 + \perp P$	h_1	
	$ADC(I_\emptyset;\ \perp\omega^{15}/I; a)$	h_2	
	$\longrightarrow n_3$	h_3	
DIV------>	$s_1 \longleftarrow B_\emptyset$	h_4	
	$MAC^1(a, f; C)$	h_5	
	$s_2 \longleftarrow C_\emptyset$	h_6	
	$B,A \longleftarrow \top/\perp(B,A); B_\emptyset; 2^{32} - \perp(B,A)/$	h_7	
	$C \longleftarrow \top/\perp C; C_\emptyset; 2^{16} - \perp C/$	h_8	
	$1 : 0 \longleftarrow (\emptyset = C)$	h_9	$\xrightarrow{=}$
	$x \longleftarrow (\perp C) \mid \perp(B,A)$	h_{10}	
$\ddagger$---	$1 : 0 - (((\ (B,A) - x) \div (\perp C)) > 2^{15})$	h_{11}	
	$B,A \longleftarrow ?(32)$	h_{12}	$\longrightarrow$
--->	$t \longleftarrow ((\perp(B,A)) - x) \div (\perp C)$	h_{13}	
	$x \longleftarrow /x; (s_1 \vee s_2); 2^{16} - x/$	h_{14}	
	$B - (16)\top x$	h_{15}	
	$A - \top/t, (s_1 \oplus s_2); 2^{16} - t/$	h_{16}	$\longrightarrow$
MPY ------->	$MAC^1(a, f{:}C)$	h_{17}	
	$s_1 \longleftarrow A_\emptyset$	h_{18}	
	$s_2 \longleftarrow C_\emptyset$	h_{19}	
	$x \longleftarrow /\perp A; A_\emptyset; 2^{16} - \perp A/$	h_{20}	
	$t \longleftarrow /\perp C; C_\emptyset; 2^{16} - \perp C/$	h_{21}	
	$B.A \longleftarrow \top/x \times t; s_1 \oplus s_2; 2^{32} - x \times t/$	h_{22}	$\longrightarrow$
DLD DST -------->	$MAC^1(a, (f,s)_{n_\emptyset}; A)$	h_{23}	
	$MAC^1(a+1, (f,s)n_\emptyset; B)$	h_{24}	

Figure 9. (Continued)

	$t \leftarrow \perp/(A,\alpha^8/B);A_\emptyset;2^{24}-(A,\alpha^8/B)/\div 2^{23}$	i_0
	$t \leftarrow t \times 2^{/\alpha^7/\omega^8/B;B_{15};(\perp\alpha^7/\omega^8/B)-2^7/}$	i_1
	$-(i_3,i_9,i_{14})_{n_\emptyset}$	i_2
FIX-------->	$t \leftarrow (\emptyset,1)_{(t<6)}+\lfloor t$	i_3
	$s_2 \leftarrow (/\alpha^7/\omega^8/B;B_{15};2^7-\perp\alpha^7/\omega^8/B/\geq 16)$	i_4
	$t \leftarrow (t,32767)_{s_2}$	i_5
	$0 \leftarrow s_2$	i_6
	$A \leftarrow \top/t;t<\emptyset;2^{16}+t/$	i_7 →
FLT--------	$b \leftarrow ((\omega^{15}/A)/\underset{\sim}{\emptyset})\emptyset$	i_8
	$\omega^{15}/A \leftarrow b\uparrow\omega^{15}/A$	i_9
	$B \leftarrow \bar{\epsilon}(16)$	i_{10}
	$\alpha^7/\omega^8/B \leftarrow \top(15-b)$	i_{11} →
FAD FSB / FMP, FDV →	$MAC^1(\perp P,f;I)$	i_{12}
	$P - \top 1 + \perp P$	i_{13}
	$ADC(I_\emptyset;\perp\omega^{15}/I;a)$	i_{14}
	$MAC(a,f;C)$	i_{15}
	$MAC(a+1,f;g)$	i_{16}
	$x \leftarrow /\perp(C,\alpha^8/g);C_\emptyset;2^{24}-\perp(C,\alpha^8/g)/\div 2^{23}$	i_{17}
	$s_2 \leftarrow /\perp\alpha^7/\omega^8/g;g_{15};(\perp\alpha^7/\omega^8/g)-2^7/$	i_{18}
	$x \leftarrow x \times 2$	i_{19}
	$b \leftarrow x(+,-,\times,\div)_{n1}\ t$	i_{20}
	$m \leftarrow (bC-2^{-127})\vee(b>(1-2^{-23})\times 2^{127})$	$i_{20.1}$

Figure 9 (Continued)

$b \leftarrow (b,\ (1-2^{-23}) \times 2^{-129})_m$	i_{21}
$b \leftarrow (b,1)_{(b>(1+2^{-22})\times(-2^{129}))\wedge(b<2^{-129})}$	i_{22}
$0 \leftarrow (b<-2^{127}) \vee (b>(1-2)^{-23}\times 127)$ $\vee((b>(1+2)^{-22}(-2^{129}))(b<2^{-129}))$	i_{23}
$t \leftarrow \emptyset$	i_{24}
$s_1 \leftarrow (b<\emptyset)$	i_{25}
$b \leftarrow \mid b$	i_{26}
$b : 1$	i_{27}
$b \leftarrow b \div 2$	i_{28}
$t \leftarrow t+1$	i_{29}
$b : \emptyset.5$	i_{30}
$b \leftarrow b \times 2$	i_{31}
$t \leftarrow t-1$	i_{32}
$B_{15},\ \alpha^7/\omega^8/B\ -\ \tau/t;(t<\emptyset);t+2^8/$	i_{33}
$b \leftarrow b \times 2^{24}$	i_{34}
$A,\alpha^8/B \leftarrow \tau/b;s_1;2^{24}\ -b/$	i_{35}

Figure 9. (Continued)

Lines b_0-b_{17}

All IRI instructions are executed here. ADX, ADY, LDX, LDY, STX, and STY refer to certain memory locations whose addresses are defined in the word following the instruction word; thus some memory access and effective address computation tasks must be done(60-63) prior to the execution of the instructions. All the other instructions of IRI do not require those tasks and enter the routine at line b_4 to skip the unnecessary steps.

Lines C_0-C_5

The EAR instruction sets are executed here. Each instruction enters at a different line.

Lines d_0-d_{21}

All the IOI instructions are executed here. The I/O devices are interfaced with the processor by these instructions; symbols V, F, and Z are used here to represent the control bits, I/O flag bits, and data buffers of all the I/O devices. Each indexed symbol refers to a specific device.

Lines e_0-e_{18}

All the S/R instructions are executed here. Each S/R instruction consists of four microinstructions. Each microinstruction is chosen from its own microinstruction set. The first microinstruction set is the same as the fourth microinstruction set for S/R instructions. The instruction execution is divided into three parts. The first part (lines e_0-e_{12}) executes the first microinstruction, the second part (lines 13-14) executes the second microinstruction, and the third part (lines 15-17) executes the third microinstruction. The fourth microinstruction is executed in the first part after the previous three microinstructions are all executed. Every S/R instruction must go through these four steps to complete the instruction execution.

Lines f_0-f_{12}

All the A/S instructions are executed here. Each A/S instruction consists of 8 microinstructions. Thus the instruction execution is divided into 8 parts, each part executing one microinstruction. Every A/S instruction must go through these 8 parts to complete the instruction execution.

Lines g_0-g_5

The JUMP instructions JLY and JPY are executed here. A memory access must be made to get the destination address of the JUMP instruction.

Lines h_0-h_{23}

All the EAMR instructions are executed here. Each of the four EAMR instructions requires two words of memory: one for the instruction code and one for the operand address. Thus at line h_0, the second memory word (operand address) is incremented by 1 to point to the next instruction. The overflow bit is set when the DIV instruction is executed if the divisor is zero or too small. In the former case (division by zero), the division will not be attempted and the B- and

A-register contents will be unchanged except that a negative quantity will be made positive. In the latter case (divisor too small), the execution will be attempted with unpredictable results left in the B- and A-registers.

Lines i_0-i_{24}

All the FPI instructions are executed here. Four of the FPI instructions are floating point arithmetic instructions which require two words of memory: one for the instruction code and one for the operand address. Since a full 15 bits are available for the operand address, these instructions can directly address any location in memory.

The execution of WMI, BIMI, BYMI, and DMI instructions is not included in the APL description here because they are not used and have nothing to do with this paper.

Microprogramming

Conventional Control Section

In a conventional computer control section, the functions performed by the instruction set determine the specified hardware design. The major advantage of this specially designed hardware is speed of instruction execution. The major disadvantage is the loss of flexibility for special applications or for enhancements. Any changes and additions to existing capabilities require changes and additions to hardware components. This is no problem for a conventional computer is there are no new machine instructions required. "The hardware has been designed to minimize timing for the instruction set" (6).

However, a computer manufacturer rarely produces an instruction set that meets the requirements of all potential users. "Hence, the manufacturer must either focus his attention on one group of users or widen his scope and generalize the hardware design to meet the needs of a number of user groups. In the latter case, the user must modify his discipline to some extent to meet the limitations of his hardware"(6).

Microprogrammed Control Section

"In the microprogrammed computer, all distinct logical functions are separated from the sequence in which those functions are performed" (6). Thus, hardware redundancy is reduced. The control store holds the microinstruction which defines the logical functions. Each machine instruction in Main Memory is performed by a sequence of microinstructions in Control Store. This sequence of microinstructions called a microprogram and is often referred as firmware. Software can be executed much faster with the application of microprogramming. This speed is achieved by two factors:

1. The memory access time of Control Store is less than that of Main Memory.
2. The microinstruction has more flexibility than the normal machine instruction.

In fact, the HP21MX Control Store where microinstruction reside, cycles more than twice as fast as Main Memory where normal machine instructions reside. In addition, microinstruction have the ability to access many internal registers and some logical functions that Main Memory programs do not have.

For example, the HP21MX floating point software subroutines were identified as very time consuming. They were microprogrammed by Hewlett-Packard and made available in ROM to users. Implementation of floating point firmware requires no change to user programs. The microprogrammed floating point instructions run about 20 times faster than the corresponding software subroutines.

As in the floating point microprogram, the user can study his software, determine the most time consuming function performed, and then microprogram these functions, that is, execute them in control store using a single memory instruction instead of a sequence of Main Memory instructions. Any software that uses these microprogrammed functions will execute at a higher speed.

The Microprogrammable Computer

Functionally, a computer consists of four major sections:

Control

Main Memory

Input and Output

Arithmetic and Logic

Each section executes under the direction of the control section by means of a microprogram. The control section reads the user's program stored in Main Memory and directs the appropriate hardware in each of the other sections.

Control Section

The control section fetches an instruction from a certain location in memory, which is specified by the Memory Register (MR), and stores it into the Instruction Register (IR), as shown in Figure 10. An

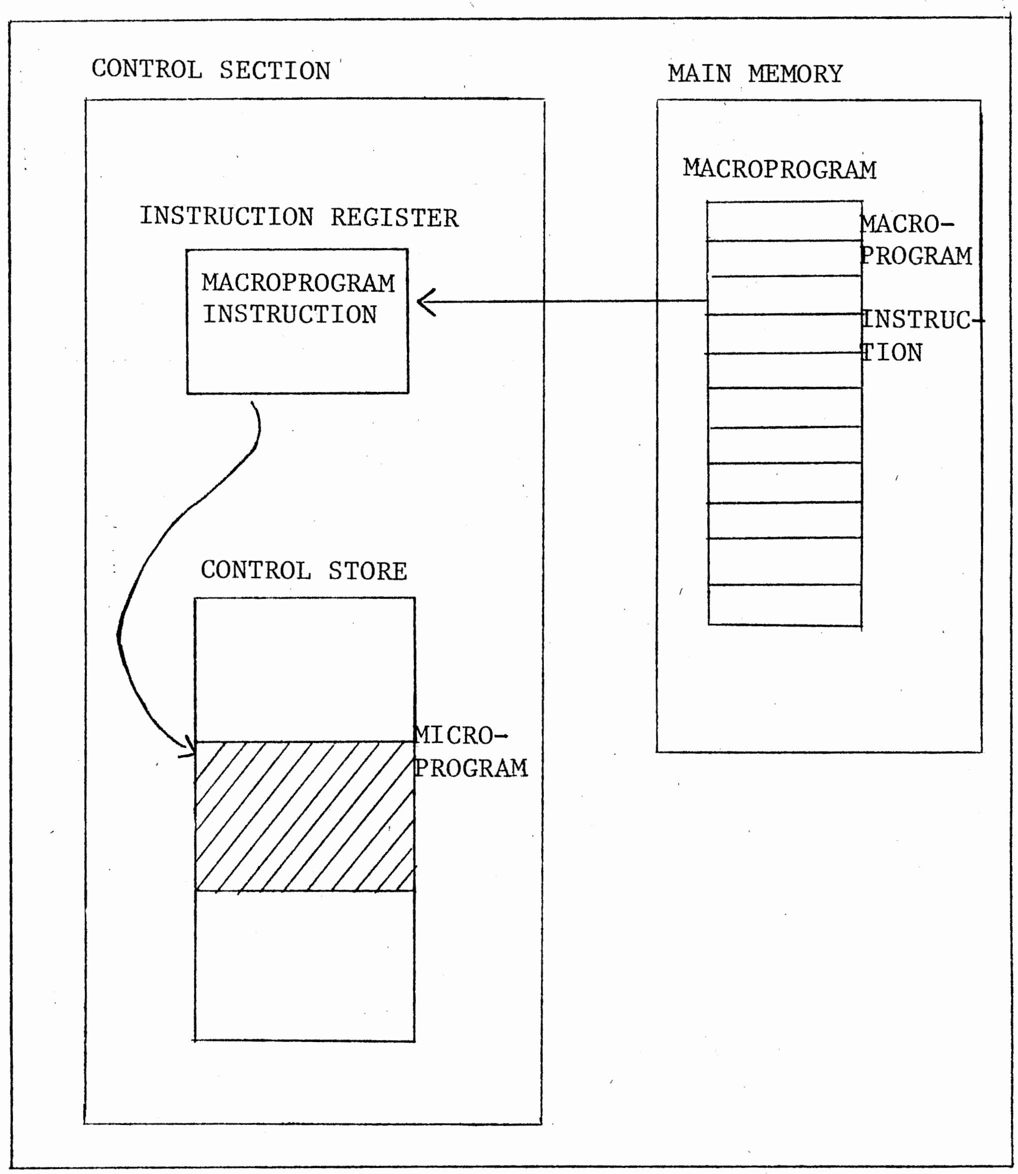

Figure 10. A Microprogram Implemention of One Macroprogram Instruction.

appropriate microprogram is determined by the IR. Conceptually, each program instruction in Main Memory is a jump to a microprogrammed routine which resides in Control Store.

The storage area for those microprograms is Control Store which may be either a Read Only Memory (ROM) or Writable Control Store (WCS). The control section that executes microprograms from ROM is referred as a Control Processor.

The Control Processor

A microprogram in the Control Processor is in command of the computer at all times. A microprogram takes program instructions from Main Memory and stores them into the IR. The upper eight bits of the IR determine the microprogram address within one of the following groups:

Basic instruction set

Extended instruction group

Floating point instruction group

User microprogram group

The basic instruction set microprogram can be regarded as a supervisor microprogram that determines when a user microprogram is called and then passes control to the user microprogram.

When a microprogram has run to completion, it returns to location 0 in Control Store (basic instruction set), returning control to the supervisor microprogram, after which the next instruction is fetched from Main Memory and stored into the IR. Successive microinstruction address are determined in the following way. The ROM Address Register

(RAR) is incremented at the start of execution of each microinstruction. When a jump is executed, the RAR is loaded with the jump target address. When a jump to a subroutine is executed, the RAR is stored into the Save Register. When a return from a subroutine is executed (RTN), the Save Register contents are transferred into RAR and the Save Register is cleared. Thus at the completion of execution of each microinstruction, the RAR holds the address of the next microinstruction.

The central data transfer path is the S-bus. The contents of all registers except the following can be directed onto the S-bus: L-register, RAR,SAVE Register, Extend Register, and the Overflow Register. The following registers can receive data from the S-bus: M-Register, T-Register, L-Register, Counter-Register, Display-Register, Display Indicator, and Instruction Register.

The T-but receives data only from the Rotate/Shifter (R/S) but can pass data to the following registers: A-Register, B-Register, Scratch Pad Register (S_1 through S_{12}), X-Register, Y-Register, P-Register, and S-Register, (Front Panel Switch Register).

The I/0-bus serves to transfer data to and from external devices under program control. In the functional block diagram (Appendix A) all the data paths are shown by the arrows. For example, the B-Register contents can be sent to S-bus and hence to the M-Register. However, the contents of the B-Register cannot be sent to S12 (Scratch Pad 12) without passing through the ALU.

Main Memory

The M-register is a 15-bit register which holds memory addresses for reading from or writing into Main Memory. Upon storing from the

M-Register, bit 15 is clear (0). The T-Register or transfer register holds the data being transferred to or from memory. The contents of both of these registers are transferred to and from the -bus. Four loader ROMS, selectable by Instruction Register bits 15 and 14, can each contain a 64-word Main Memory program which may be loaded into Main Memory and used to load Main Memory from a peripheral device, or to perform any other function desired by the user.

Two flags are associated with memory: the A-Register Addressable Flag (AAF) and the B-Register Addressable Flag (BAF). These flags are required to allow the A- and B-registers to be addressed as locations 0 and 1, respectively, of Main Memory.

Input and Output

The Central Interrupt Register (CIR) is a 6-bit register associated with the I/0 interrupt circuitry. It is loaded with the select code of the interrupting device under program control and passed to the S-bus. Whenever the CIR is loaded, and Interrupt Acknowledge (IAK) signal is issued to the I/0 device. The I/0 bus transfers data to and from external devices. Two flags are associated with I/0: the interrupt pending flag and the I/0 skip condition met flag. The Interrupt Enable Register is used to disable or enable the recognition of all interrupts, except Memory protect, parity, and power failure interrupts.

Arithmetic and Logic Section

This section consists of the Arithmetic and Logic Unit (ALU), the twenty-two Rotate/Shifter (R/S) registers, and six flags.

The ALU and R/S are the only units that execute functional modifications on the data. The ALU receives inDut from the S-bus and from the L-register (Latch Register). Output from the ALU goes to the R/S which places its output on the T-bus.

Output from the ALU and R/S can be stored in one of the folloiwng registers via the T-bus: A-Register, B-Register, Scratch Pad Registers (S_1 through S_{12}), X-Register, Y-Register, P-Register, and S-Register.

Recall that the P-register holds the macroprogram (main memory) address. The P-register must be under control of the microprogram which must insure that it contains the proper address after the micro-program is complete. When the microprogram is complete, the resulting P-Register value is the address of the next macroinstruction to be executed. Note that the Basic Instruction Set fetch routine (at Control Store address 0) automatically increments the P-Register after the macroinstruction is fetched. Thus for one-word user macro-instruction function codes, no further incrementing of the P-Register is necessary in the user microprogram.

The S-Register is reserved for internal storage of the Front Panel Switch Register. Note that all of those registers can also be sent along the S-bus for storage into memory, passage to an external device, or input to the ALU.

The Extend Register is a one-bit register used in shift operations to link the A- and B-Registers or to indicate a "carry" arithmetic result out of the A- or B-Registers. The overflow is a one-bit register used to indicate an arithmetic overflow from the ALU. These two registers can also be used as flags.

Implementation of a Polynomial Algorithm in the HP21MX Computer

The four tasks which are illustrated in Chapter I are performed in this chapter. One of them is to program the polynomial algorithm in Hp 21 assembler language for evaluating the sine function. The other task does the same thing but uses a microprogram instead of the program coded in assembler language.

The particular polynomial algorithm used for evaluating sine functions has been determined in Chapter II and is shown as follows:

$$\sin x = c_1 x + c_3 x^3 + c_5 x^5 \tag{4.1}$$

where

$$c_1 = 1.5706268$$

$$c_3 = -0.6432292$$

$$c_5 = 0.0727105$$

$$-1 \le x \le 1$$

For evaluating the sine of an angle θ, x is substituted with $2\theta/\Pi$ in Eq. (4.1); then $\sin\theta$ can be computed by

$$\sin\theta = c_1 \left(\frac{20}{\Pi}\right) + c_3 \left(\frac{2\theta}{\Pi}\right)^3 + c_5 \left(\frac{2\theta}{\Pi}\right)^5$$

In order to reduce the execution time when implemented this algorithm in the computer, Eq. (4.1) can be factored as follows:

$$\sin\frac{\Pi}{2}x = x(c_1 + x^2(c_3 + c_5 x^2)) \tag{4.2}$$

Although Eq. (4.1) and Eq. (4.2) give the same result in computation, they require a different number of multiplications.

Inspection of Eq. (4.1) shows that the number of multiplications required is 11, while the number of multiplications required by

Eq. (4.2) is 7. As mentioned in Chapter I, the multiplication function is one of the most time-consuming functions. Thus Eq. (4.2) definitely is more efficient than Eq. (4.1) when implemented in the computer.

For the reason mentioned above, Eq. (4.2) is used for both the microprogram and the program coded in assembly language. The results of these two implementations are listed in Tables VIII and IX. The program listings are listed in Appendix B.

Implementation of the Cordic Algorithm on the HP21MX Computer

The Cordic algorithm has been introduced in Chapter II. To use it for evaluation of the sine function, the value selected for n is a function of the desired computing accuracy. Theoretically, the larger the value of n is the more accurate the result.

Actually, it is impossible to represent a number to any degree of accuracy in any computer because the accuracy of all computers is limited by the number of bits in a word. In the HP21MX computer, there are 16 bits in a word. When the Cordic algorithm is used to evaluate the sine function, the value of n not only determines the accuracy of the result, but also affects the execution time of the program. There is a trade-off between accuracy and execution time; i.e., when n increases, the accuracy is increased as is the execution time. In order to get the greatest accuracy and the least execution time, the optimum value of n must be found. As discussed in Chapter II, a set of ATR constants, α_i, i=1,...,n,, can be obtained from Eq. (4.3).

$$\alpha_i = \tan^{-1} 2^{-(i-2)} \quad \text{for } 2 \le i \le n \tag{4.3}$$

TABLE VIII

POLYNOMIAL METHOD IMPLEMENTATION RESULTS (ASSEMBLY LANGUAGE) OF EVALUATING THE SINE FUNCTION

Angle(Radians)	Sin	Execution Time(Mili-Sec)
-1.5	-0.997558	0.081
-1.4	-0.985351	0.081
-1.3	-0.963378	0.081
-1.2	-0.932128	0.081
-1.1	-0.891357	0.081
-1.0	-0.841552	0.081
-0.9	-0.783447	0.081
-0.8	-0.717285	0.081
-0.7	-0.644287	0.081
-0.6	-0.564697	0.081
-0.5	-0.479492	0.081
-0.4	-0.389404	0.081
-0.3	-0.295654	0.081
-0.2	-0.198730	0.081
-0.1	-0.099853	0.081
0.0	0.0	0.081
0.1	0.099609	0.081
0.2	0.198486	0.081
0.3	0.295410	0.081
0.4	0.389160	0.081

TABLE VIII (Continued)

Angle(Radians)	Sin	Execution Time(Mili-Sec)
0.5	0.564453	0.049
0.6	0.564453	0.049
0.7	0.644042	0.049
0.8	0.717041	0.049
0.9	0.783203	0.049
0.1	0.841308	0.049
1.1	0.891113	0.049
1.2	0.931884	0.049
1.3	0.963134	0.049
1.4	0.985107	0.049
1.5	0.997314	0.049

TABLE IX

POLYNOMIAL METHOD IMPLEMENTATION RESULTS (MICROPROGRAM) OF EVALUATING THE SINE FUNCTION

Angle(Radians)	Sin	Execution Time(Mili-Sec)
-1.5	-0.997558	0.049
-1.4	-0.984351	0.049
-1.3	-0.963378	0.049
-1.2	-0.932128	0.049
-1.1	-0.891357	0.049
-1.0	-0.841552	0.049
-0.9	-0.783447	0.049
-0.8	-0.717285	0.049
-0.7	-0.644287	0.049
-0.6	-0.564697	0.049
-0.5	-0.479492	0.049
-0.4	-0.389494	0.049
-0.3	-0.295654	0.049
-0.2	-0.198730	0.049
-0.1	-0.099353	0.049
0.0	0.0	0.049
0.1	0.099609	0.049
0.2	0.198486	0.049
0.3	0.295410	0.049
0.4	0.389160	0.049

TABLE IX (Continued)

Angle(Radians)	Sin	Execution Time(Mili-Sec)
0.5	0.479248	0.081
0.6	0.564453	0.081
0.7	0.644042	0.081
0.8	0.717041	0.081
0.9	0.783203	0.081
1.0	0.841308	0.081
1.1	0.891113	0.081
1.2	0.931884	0.081
1.3	0.963134	0.081
1.4	0.985107	0.081
1.5	0.997314	0.081

When implementing the Cordic algorithm in the HP21MX computer, α_i will be divided by 180° and then represented in 16 binary digits. For example, if α_1 = 90°, then 90°/180° = 0.5_{10} = 0.40000_8. 040000_8 will be stored in the computer. If Eq. (4.3) is used to find the ATR constants n=1 to n=16, the values of α_i are: α_1 = 040000, α_2 = 020000, α_3 = 011344, α_4 = 004773, α_5 = 002421, α_6 = 001213, α_7 = 000505, α_8 = 000242, α_9 = 0001212, α_{10} = 000050, α_{11} = 000024, α_{12} = 000012, α_{13} = 000005, α_{14} = 000002, α_{15} = 000001, α_{16} = 000000.

Because the ATR constant is represented with a 16-bit word in the HP21MX computer, when n > 15, the constant will be too small to be represented. Thus the value 15 is the best choice for the value of n. This yields the most accurate result without excessive execution time.

Once the value of n is determined, the value of k can be found as well. The formula to obtain the constant k is:

$$k = 1+2 \quad 1+2^{-2} \qquad 1+2^{-2(n-2)} \tag{4.5}$$

When the constant k is computed by Eq. (4.5) with n = 15, the result is:

$$k = 1.646744$$

The original coordinate vector in the Cordic algorithm is:

$$V = 1/k = 0.6072589$$

One critical problem occurs immediately when the Cordic algorithm is being implemented in the HP21MX computer. Review of the Cordic machine in Chapter III shows that the best feature of Cordic which speeds up computation is that it has three adder-subtractors which can operate simultaneously. In the HP21MX computer, although there are two registers (A and B) which can operate like an adder-subtractor in Cordic, they cannot operate simultaneously. Due to this hardware limitation, the only way to simulate these parallel adder-subtractor operations is to execute sequentially.

The flowchart for the assembler program which simulates the Cordic algorithm in the HP21MX computer is shown in Figure 11.

An AHPL description for the microprogram which emulates the Cordic algorithm in the HP21MX computer is shown in Figure 12.

Both program listings are shown in Appendix B. The programming results for these two implementations are listed in Tables X and XI.

Calculation of Execution Time

To calculate the execution time of both the macroprogram and the microprogram, the Time Base Generator (TBG) and interrupt feature are used. The TBG generates an interrupt signal for a specified time interval; the CPU acknowledges the interrupt and forces the current computer program to suspend and transfer control to a service subroutine which records the number of times that the clock interrupt has occurred. At the end of program, the program execution time can be calculated from the following equation:

$$T = \frac{N \times TI}{L} \quad \text{where}$$

T = program execution time

N = number of clock interrupts

TI = interrupt time interval of Time Base Generator

L = number of times that the program has been executed

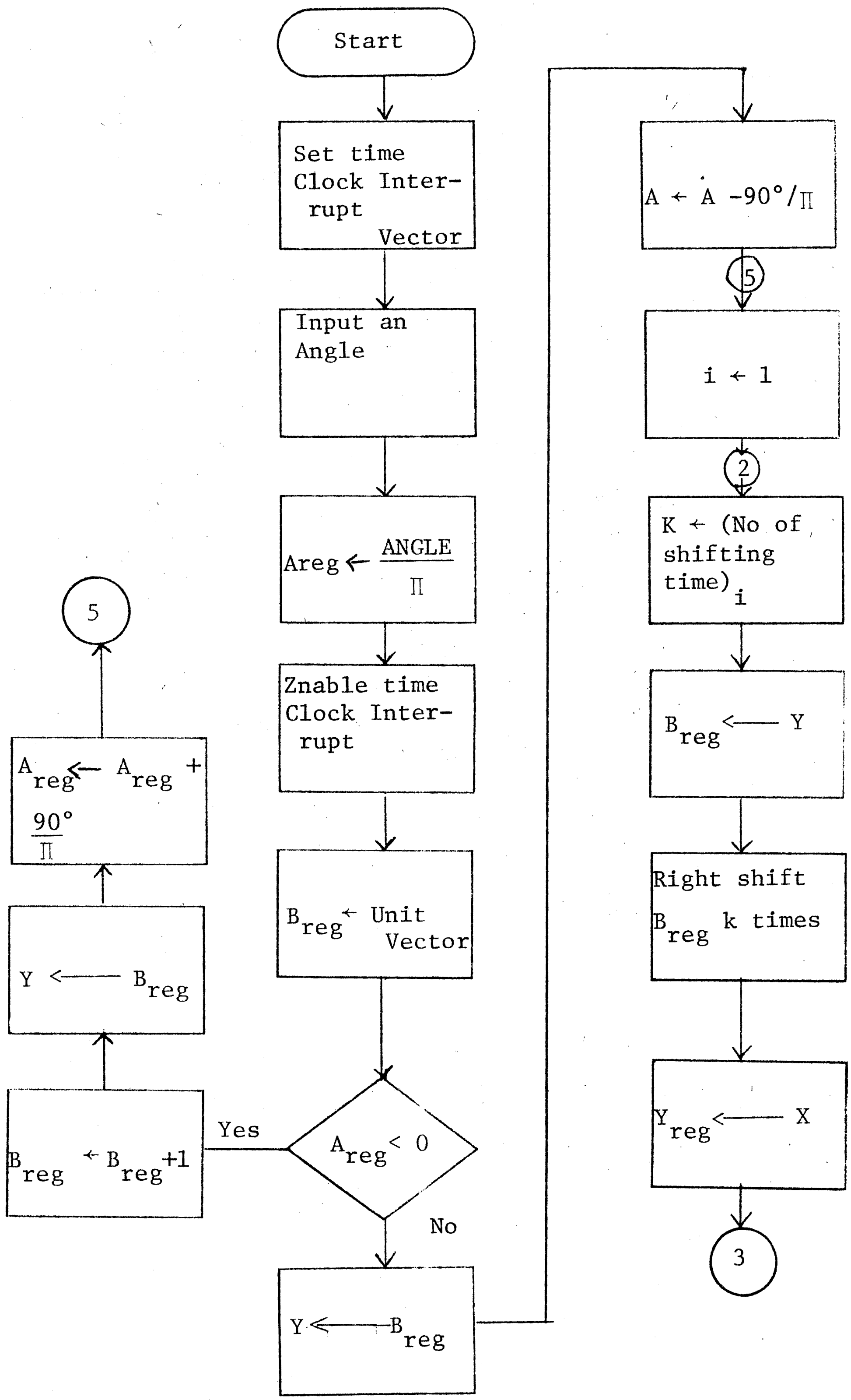

Figure 11. Cordic Algorithm

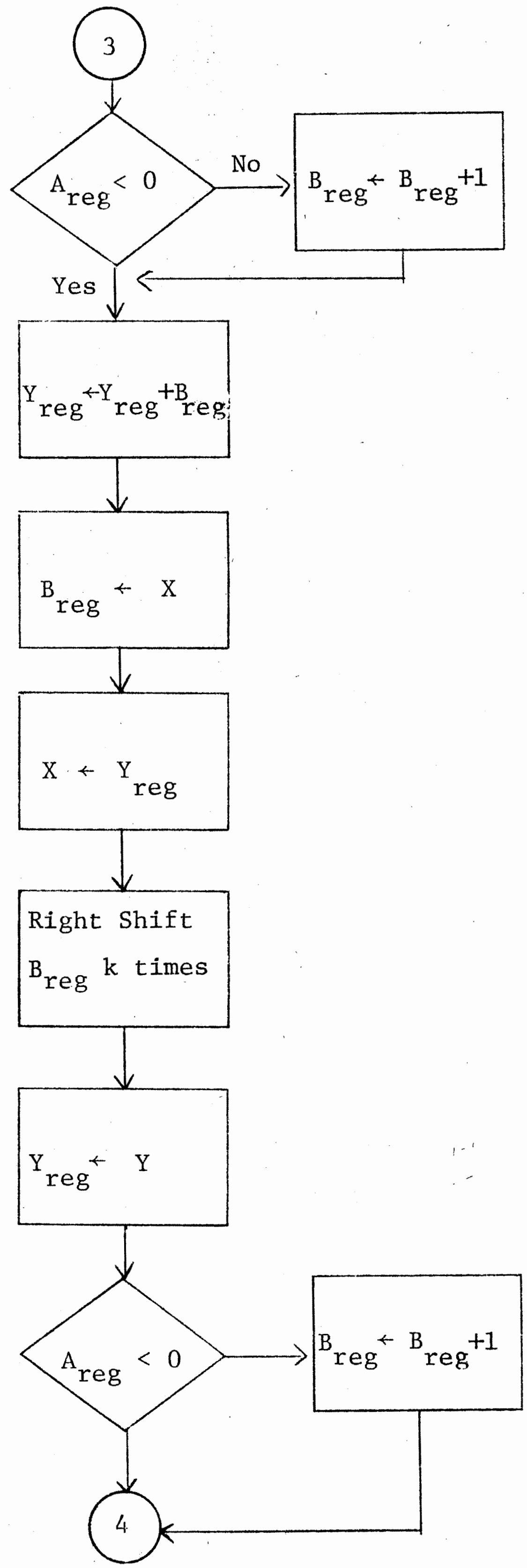

Figure 11. (Continued)

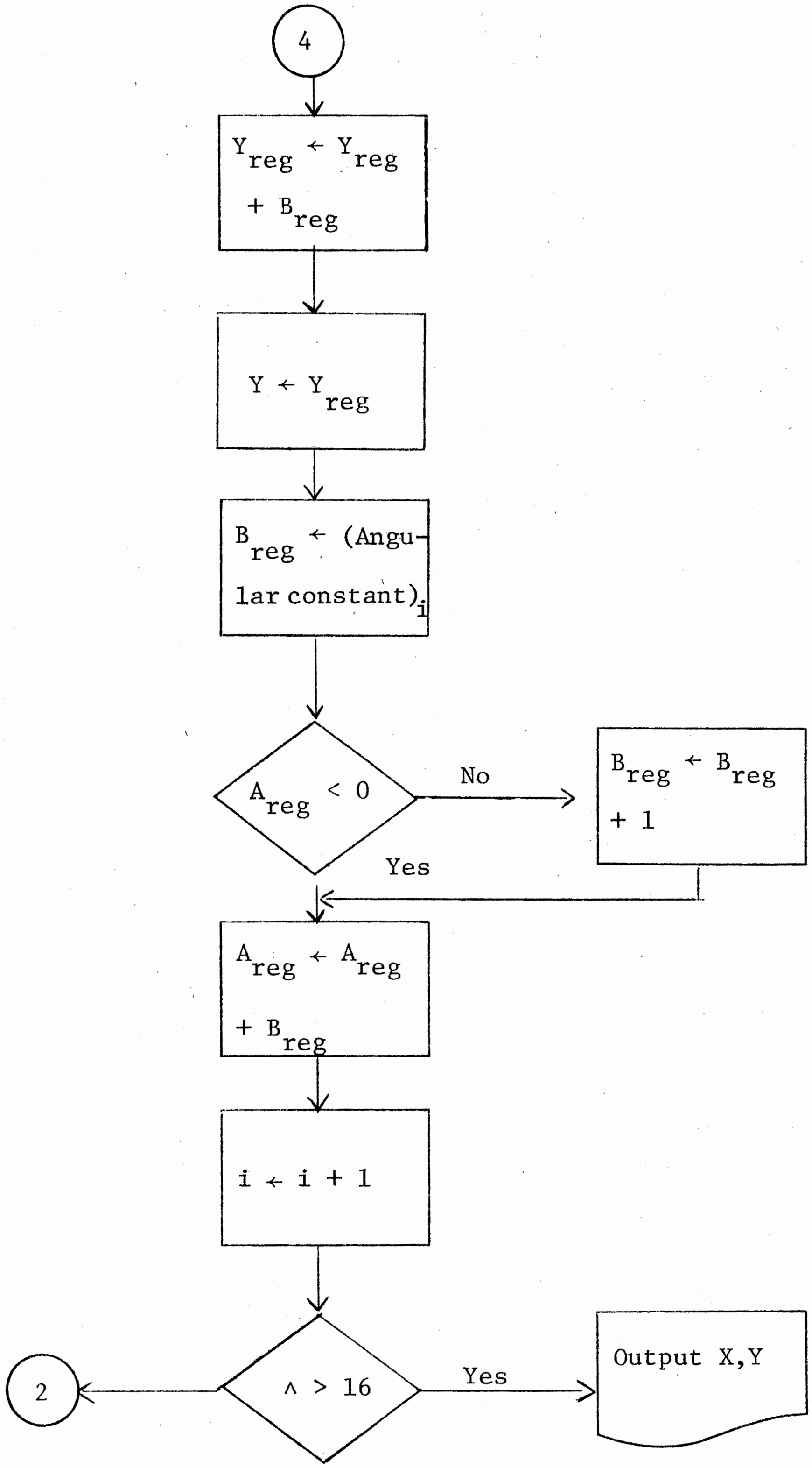

Figure 11. (Continued)

0 $MR \leftarrow P$

1 $P \leftarrow \top 1 + \bot P$

2 $MAC^{1}(\bot MR, f; T)$

3 $X \leftarrow T$

4 $MR \leftarrow P$

5 $P \leftarrow \top 1 + \bot P$

6 $MAC^{1}(\bot MR, f, T)$

7 $S7 \leftarrow T$

8 $S6 \leftarrow A$

9 $\rightarrow (10,14)_{A0}$

10 $S7 \leftarrow \overline{S7}$

11 $S7 \leftarrow (16)\top 1 + \bot S7$

12 $X \leftarrow \overline{X}$

13 $X \leftarrow (16)\top 1 + \bot X$

14 $Y \leftarrow X$

15 $X \leftarrow \overline{8}(16)$

16 $E, S6 \leftarrow \top(\bot S6) + (\bot S7)$

17 $S4 \leftarrow S^{\overline{12,13,14}}(16)$

18 $S3 \leftarrow \overline{8}(16)$

19 $S5 \leftarrow X$

20 $L \leftarrow Y$

21 $\rightarrow 37$

22 $CTR \leftarrow \omega^{8}/S3$

23 $B \leftarrow X$

Figure 12. The AHPL Description for the Cordic Algorithm in Implementation in HP21MX Microprogram

```
24    (v/CTR):0,(=,≠) → (29,25)
25    B ← ↑o B
26    (∧/CTR):1,(=,≠) → (29,27)
27    CTR ← ⊤1+⊥CTR
28    → 25
29    S5 ← B
30    CTR ← ω^8/S3
31    B ← Y
32    (V/CTR):0,(=,≠) → (37,33)
33    B ← ↑o B
34    (∧/CTR):1,(=,≠) → (37,35)
35    CTR ← ⊤1+⊥CTR
36    → 33
37    MR ← P
38    P ← ⊤1+⊥P
39    MAC^1(⊥MR,f;T)
40    S7 ← T
41    → (42,48)_(S6)0
42    E,X ← (17) (⊥X)+2^16-⊥L
43    L ← S5
44    E,Y ← (17) (⊥Y)+(⊥L)
45    L ← S7
46    E,S6 ← (17) (⊥S6)+2^16-⊥L
47    → 53
48    E,X ← (17) (⊥X)+(⊥L)
```

Figure 12. (Continued)

49 $L \leftarrow S5$

50 $E,Y \leftarrow (17)\top(\bot Y)+2^{16}-\bot L$

51 $L \leftarrow S7$

52 $E,S6 \leftarrow (17)\top(\bot S6)+\bot L$

53 $E,S4 \leftarrow (17)\top 1+\bot S4$

54 $\rightarrow (57,55)_{(\vee/S4)}$

55 $E,S3 \leftarrow (17)\top(\bot S6)+2^{16}-1$

56 $\rightarrow 22$

57 RETURN TO MACROPRAM

Figure 12. (Continued)

TABLE X

CORDIC ALGORITHM IMPLEMENTATION RESULTS
(ASSEMBLY LANGUAGE) OF EVALUATING
THE SINE FUNCTION

Angle(Radians)	Sin	Execution Time(Mili-Sec)
0.0	0.000244	3.304
0.1	0.099975	3.297
0.2	0.198913	3.310
0.3	0.295410	3.313
0.4	0.389587	3.300
0.5	0.479431	3.313
0.6	0.564758	3.305
0.7	0.644226	3.305
0.8	0.717407	3.310
0.9	0.783325	3.304
1.0	0.841369	3.305
1.1	0.891174	3.313
1.2	0.932206	3.311
1.3	0.963562	3.306
1.4	0.985351	3.311
1.5	0.997558	3.318
1.6	0.999450	3.310
1.7	0.991760	3.318
1.8	0.973693	3.313
1.9	0.946350	3.305

TABLE X (Continued)

Angle (Radians)	Sin	Execution Time (Mili-Sec)
2.0	0.909301	3.316
2.1	0.863220	3.320
2.2	0.808654	3.312
2.3	0.745666	3.310
2.4	0.675476	3.312
2.5	0.598510	3.314
2.6	0.515686	3.324
2.7	0.427795	3.311
2.8	0.334716	3.313
2.9	0.239074	3.321
3.0	0.140869	3.311
3.1	0.041564	3.316
3.2	-0.058654	3.304
3.3	-0.157592	3.311
3.4	-0.255798	3.320
3.5	-0.350646	3.317
3.6	-0.442016	3.320
3.7	-0.530090	3.327
3.8	-0.611999	3.311
3.9	-0.687622	3.314
4.0	-0.756713	3.329
4.1	-0.818054	3.302
4.2	-0.871520	3.327
4.3	-0.916320	3.321
4.4	-0.951599	3.311

TABLE X (Continued)

Angle(Radians)	Sin	Execution Time(Mili-Sec)
4.5	-0.977539	3.322
4.6	-0.999877	3.314
4.7	-0.999877	3.314
4.8	-0.996032	3.310
4.9	-0.982422	3.312
5.0	-0.958801	3.314
5.1	-0.925901	3.311
5.2	-0.883422	3.314
5.3	-0.832153	3.308
5.4	-0.772583	3.304
5.5	-0.705505	3.309
5.6	-0.631530	3.303
5.7	-0.550659	3.309
5.8	-0.464599	3.310
5.9	-0.373840	3.305
6.0	-0.279541	3.319
6.1	-0.182312	3.314
6.2	-0.082885	3.312

TABLE XI

CORDIC ALGORITHM IMPLEMENTATION RESULTS (MICROPROGRAM) OF EVALUATING THE SINE FUNCTION

Angle(Radians)	Sin	Execution Time(Mili-Sec)
0.0	0.000244	0.105
0.1	0.099975	0.126
0.2	0.198913	0.112
0.3	0.295410	0.108
0.4	0.389587	0.106
0.5	0.479431	0.104
0.6	0.564758	0.108
0.7	0.644226	0.107
0.8	0.717407	0.113
0.9	0.783325	0.097
1.0	0.841369	0.110
1.1	0.891174	0.111
1.2	0.932206	0.114
1.3	0.963562	0.109
1.4	0.985351	0.104
1.5	0.997558	0.105
1.6	0.999450	0.104
1.7	0.991760	0.105
1.8	0.973693	0.114
1.9	0.946350	0.106

TABLE XI (Continued)

Angle(Radians)	Sin	Execution Time(Mili-Sec)
2.0	0.909301	0.111
2.1	0.863220	0.105
2.2	0.808654	0.106
2.3	0.745666	0.105
2.4	0.675476	0.116
2.5	0.598510	0.111
2.6	0.515686	0.107
2.7	0.427795	0.116
2.8	0.334716	0.107
2.9	0.239074	0.114
3.0	0.140869	0.102
3.1	0.041564	0.103
3.2	-0.058654	0.101
3.3	-0.157592	0.105
3.4	-0.255798	0.106
3.5	-0.350646	0.112
3.6	-0.442016	0.110
3.7	-0.530090	0.105
3.8	-0.611999	0.109
3.9	-0.687622	0.097
4.0	-0.756713	0.108
4.1	-0.818054	0.112
4.2	-0.871520	0.107
4.3	-0.916320	0.107
4.4	-0.951599	0.111

TABLE XI (Continued)

Angle(Radians)	Sin	Execution Time(Mili-Sec)
4.5	-0.977539	0.106
4.6	-0.993774	0.107
4.7	-0.999877	0.107
4.8	-0.996032	0.107
4.9	-0.982422	0.102
5.0	-0.982422	0.102
5.1	-0.925901	0.101
5.2	-0.883422	0.110
5.3	-0.832153	0.108
5.4	-0.772483	0.111
5.5	-0.705505	0.110
5.6	-0.631530	0.115
5.7	-0.550659	0.116
5.8	-0.464599	0.107
5.9	-0.373840	0.114
6.0	-0.279541	0.111
6.1	-0.182312	0.110
6.2	-0.082885	0.107

CHAPTER V

OTHER USES OF CORDIC

The Cordic algorithm may also be applied in solving many other mathematic problems as well as being applied in the evaluation of the sine and cosine functions. Decimal to binary and binary to decimal conversion, arctangent function computation, fourier transformation, et.al., can be done by the Cordic algorithm--a different way from the conventional methods. Arctangent function computation and decimal to binary conversions are chosen in this chapter to demonstrate how the Cordic algorithm is applied to solve these problems.

Arctangent Algorithm

This algorithm is obtained by reversing the sine and cosine algorithms. In this algorithm, the value V which equals Y/X is known (X and Y are components of a vector.) The vector is rotated with respect to the positive X-axis. The angle traversed is the angle whose tangent equals Y/X.

Functional Description

The VECTORING mode is used in this application. To illustrate the details of this algorithm, Figure 2 in Chapter III is referred to again. The value of v is checked before the initialization of the X- and Y-registers. If the value of v is greater than 1 then the Y-register is initialized with 1 and the X-register is initialized with v; otherwise the X-register is initialized with 1 and the Y-register is initialized with v. The Angle Register (A-register) is always initialized with 0. A sign digit of 0 in the Y-register establishes a v_i of -1, which causes the top adder-subtractor to be set to subtract and the middle and bottom adder-subtractors to add. A sign digit of 1 has the opposite effect. The ATR constants are the same as those used in Chapter III. The VECTORING computing sequence as described in Table II is started. The angle whose tangent equals to v is taken from the A-register after the final computation step.

Decimal to Binary Conversions in Cordic

A technique is formulated for using the Cordic arithmetic unit to convert between angles expressed in binary fractions of a half revolution and angles expressed in degrees and minutes in the 8421-code.

The Cordic decimal-to-binary conversion technique may be compared to a conventional conversion technique in which the 8421-code and binary arithmetic are utilized. The conventional conversion technique is based upon the 8421-code definition of the value of a decimal digit, N, located i placed to the left of the units position, as given by

$$N \times 10^i = n_4 \ (8 \times 10^i) + n_3 \ (4 \times 10^i) + n_2 \ (2 \times 10^i) + n_1 (1 \times 10^i) \tag{5.1}$$

where n_4, n_3, n_2, and n_1 are equal to zero or one. The constants 8×10^i, 4×10^i, 2×10^i, and 1×10^i, evaluated in binary for all values of i to be used, are required in the conversion. For example, 5° in 8421-code is

$$45° = (0 \times 8 \times 10 + 1 \times 4 \times 10 + 0 \times 2 \times 10 + 0 \times 1 \times 10)$$
$$+ (0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1)$$
$$45° = (0100), (0101).$$

For example, 86° can be written as

$$86° = (1 \times 8 \times 10 + 0 \times 4 \times 10 + 0 \times 2 \times 10 + 0 \times 1 \times 10) + (0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1)$$
$$86° = (1000). (0110)$$

The conversion of a negative angle is accomplished in the same way, and the result is then complemented by subtracting the binary magnitude from zero. For example, -86° is (0111)(1010) which is the 2's complement of 86°.

The binary value of 45° as a fraction of half revolution is shown in Table XII.

In Table XII at each step a binary constant is either added or not added, depending upon whether the 8421-code variable is 1 or 0, respectively. In order to use the Cordic principle, it is necessary either to add or to subtract a constant. The use of addition or subtraction is controlled by a code variable placed in the sign digit position of an arithmetic unit register. The problem of conversion by adding and subtracting constants is considered first. Subsequently, the method of properly positioning the code variables for control is presented.

By analogy to the way in which a code variable of +1 is used to establish the addition of a constant, a variable of -1 is used to establish subtraction. Therefore, it is desired that a binary code with +1 and -1 variables be used to represent decimal angles in Cordic. For convenience, the desired code is called a ± (plus-minus) code.

TABLE XII

THE CONVENTIONAL DECIMAL-TO-BINARY CONVERSION

Constants Degree	Constants-Binary Fraction of half Revolution		8421-Code Variable		Product Term
8 x 10	.01110010	x	0	=	0.00000000
4 x 10	.00111001	x	1	=	0.00111001
2 x 10	.00011100	x	0	=	0.00000000
1 x 10	.00001011	x	0	=	0.00000000
8	.00000110	x	0	=	0.00000000
4	.00000011	x	1	=	0.00000110
2	.00000011	x	0	=	0.00000000
1	.00000001	x	1	=	0.00000001

Accumulated sum = 2^{-2} half revolution = .01000000.

The 8, 4, 2, 1 weights cannot be applied directly to a four-digit $\pm$ code because all possible sums of binary-weighted $\pm$ code digits are odd. Therefore, a transformation of the decimal digits 0, 1, ..., 9, into a set of ten odd integers is necessary. The set of ten odd integers -0, -7, ..., -1, +1, ..., +9 is selected.

The equation transforming a decimal digit N, having one of the values, 0, 1, ..., 9, into a digit Y having one of the values -9, -1, ..., +9 is

$$Y = 2N - 9 \tag{5.2}$$

The equation for the inverse transformation is

$$N = \frac{1}{2}Y + \frac{9}{2} \tag{5.3}$$

Applying the factor of $\frac{1}{2}$ in (5.3) to the 8421-weight results in the $\pm$ code equation

$$N = Y_4 \cdot 4 + Y_3 \cdot 2 + Y_2 \cdot 1 + Y_1 \cdot \frac{1}{2} + C \tag{5.4}$$

where $Y_j = +1$ or -1 and $C = \frac{9}{2}$. A factor of 10^i may be applied to each term in (5.4), as was done in (5.1), account for the position of the digit N. The pattern the Y_j variables of the code of (5.4), with $C = \frac{9}{2}$ and with 0's used t · represent -1's, is identical to that of the Excess-3 code.

Equation (5.4) can be applied to each digit position, and the constant term c x 10^i for all decimal digit positions is added in binary to the accumulated sum. As an example 45° will be converted from $\pm$ (excess-3) code to binary as follows:

for 45°

$C_2 = \frac{9}{2} = 4.5$

$C_1 = \frac{9}{2} \times 10. = 45$

$C = C_1 + C_2 = 49.5 =$ total constant

Consequently the constant for 45° is 49.5.

The $\pm$ 1 code representation is

$$5 + 3 = 8 = (1000)_2 = (+---)$$

$$4 + 3 = 7 = (0111)_2 = (-+++)$$

Where each digit must be added to 3 for excess -3. The zero stands for minus one and one for plus one. Thus

$$45° = (-+++)\ (+---)$$

The complete conversion of 45° is shown in Table XIII.

Where from equation (5.4)

$$X = 4 \times 10^i Y_4 + 2\ 10^i Y_3 + 1 \times 10^i Y_2 + \frac{1}{2} \times 10^i Y_1 + C$$

$$45° = (40Y_4 + 20Y_3 + 10Y_2 + 5Y_1) + (4Y_4 + 2Y_3 + 1Y_2 + \frac{1}{2}Y_1) + C$$

$$45° = (-40 + 20 + 10 + 5) + (4 - 2 - 1 - \frac{1}{2}) + 49.5°$$

Successive digits of the $\pm$ code must control successive setting of the adder-subtractors in order for the proper sequence of additions and subtractions to occur as indicated in the previous table. The settings of the adder-subtractors during the conversion operation are established by the value of the sign digit located in the Y-register.

In positioning the $\pm$ code digits for control, the technique of nonrestoring division is useful because successive quotient digits are

TABLE XIII

DECIMAL-TO-BINARY CONVERSIONS
IN CORDIC

Constant Degrees	Constant-Bainry Fraction of Half Revolution		± Code	Product	Accumulated Sum
49.5	.0100011001110	(correction)		.010001100110	.010001100110
40	.001110001110	x	-1	= -.001110001110	.000011011000
20	.000111000111	x	+1	= +.000111000111	.001010011111
10	.0000011100100	x	+1	= +.000011100100	.001110000011
5	.00001110010	x	+1	= +.000001110010	.001111110101
4	.000001011011	x	+1	= +.00000101101	.010001010000
2	.000000101110	x	-1	= -.000000101110	.010000100010
1	.000000010111	x	-1	= -.000000010111	.010000001011
1/2	.000000001011	x	-1	= -.00000001011	.0100000000

The accumulated sum = 2^{-2} half revolution = 0.010000000000 given by the sign of successive remainders. Dividing the number representing the $\pm$ code of the angle by 1 produces the signs of successive remainders. In Cordic this is accomplished as follows:

1) If the remainder is positive, subtract the divisor.
   If the remainder is negative, add the divisor.
2) Shift the divisor one place to the right.
3) Repeat 1 and 2.

The positioning of digits of the $\pm$ code for 45° is illustrated by following the above rules as shown in Table XIV.

In decimal-to-binary conversion, the $\pm$ code for the desired angle is placed in the Y-register and the divisor of 1 is placed in the X-register. A sign digit of 0 in the Y-register establishes a Y_i of -1, which causes the top adder-subtractor, Figure 13, to subtract and the bottom adder-subtractor to add. A sign digit of 1 has the opposite effect. The constant C in (5.4) is initially placed in the angle register and successive constants are introduced into the bottom adder-subtractor as shown in Figure 13. As one step of the division is taking place to establish the next setting of the adder-subtractors, a constant is being added or subtracted to modify the quantity in the angle register according to the sign digit in the Y-register at the beginning of the step. The binary angle is taken from the bottom adder-subtractor on the final computation step.

TABLE XIV

GENERATION OF ± CODE FOR 45°

				Sign of Remainder	
(−+++)	(+−−−)	0111 1000			
sub		1			
		1111 1000	−		
add		1			
		0011 1000	+		
sub		1		= 0111	7 in excess 3
		0001 1000	+		
sub		1			
		0000 1000	+		
sub		1			
		0000 0000	+		
sub		1			
		1111 1100	−		
add		1		= 1000	8 in excess 3
		1111 1110	−		
add		1			
		1111 1111	−		

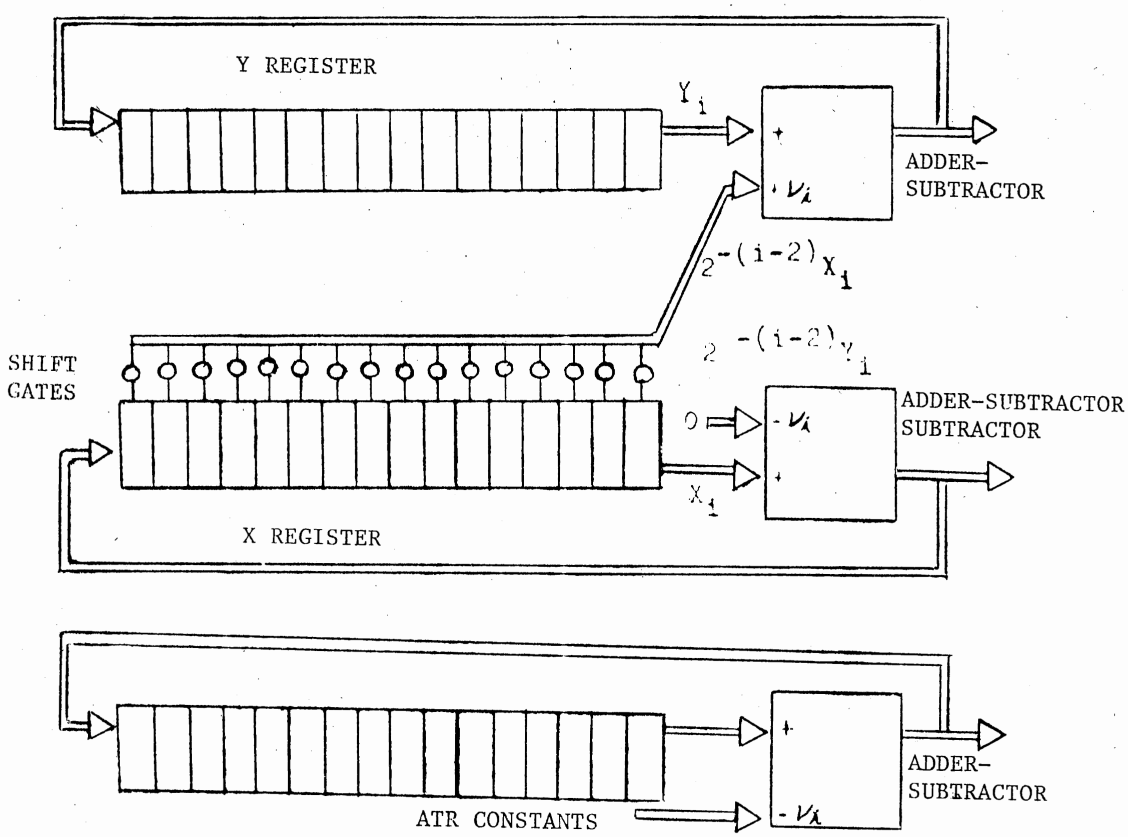

Figure 13. Implementation of $\pm$ Code to Binary Conversion.

CHAPTER VI

SUMMARY AND CONCLUSIONS

The results of the programming tasks discussed in the previous chapters are shown in Tables VIII - XI.

In order to compare the accuracy of the results obtained from each task, a set of standard sine function values is obtained. The result of each task is compared to these standard values and the accuracy is thus determined.

For the convenience of further description, the four tasks which have been accomplished in Chapter IV are designated Task 1, Task 2, Task 3 and Task 4:

Task 1 - polynomial method implemented in assembly coded program.

Task 2 - polynomial method implemented in microcode.

Task 3 - Cordic algorithm implemented in assembly coded program.

Task 4 - Cordic algorithm implemented in microcode.

Note that the sine values of Task 1 are identical to those of Task 2, while the sine values of Task 3 are identical to those of Task 4. Thus, only two sets of results are compared with the standard sine values, as shown in Tables XV and XVI. According to these tables, both tasks are accurate up to three decimal digits; in other words, all the tasks give about the same accuracy of sine values.

The execution time of each task is shown in Tables VIII -XI. By reviewing those tables it is found that Task 1 is the most time-

TABLE XV

THE COMPARISON BETWEEN THE CORDIC ALGORITHM IMPLEMENTATION RESULT AND THE STANDARD SINE VALUE

Angle(Radian)	Sin(Cordic)	Sin(Correct)	Error
0.0	0.000244	0.0	0.000244
0.1	0.099975	0.0998334	0.0001416
0.2	0.198913	0.198669	0.000244
0.3	0.295410	0.29552	0.00011
0.4	0.389487	0.389418	0.000169
0.5	0.479431	0.479425	0.000006
0.6	0.564758	0.564642	0.000116
0.7	0.644226	0.644218	0.000008
0.8	0.717407	0.717356	0.000051
0.9	0.783325	0.783327	0.000002
10.	0.841369	0.841471	0.000102
1.1	0.891174	0.891207	0.000033
1.2	0.932206	0.932039	0.000167
1.3	0.963562	0.963558	0.000004
1.4	0.985351	0.98545	0.000099
1.5	0.997558	0.997495	0.0000063
1.6	0.999450	0.999574	0.000124
1.7	0.991760	0.991665	0.000095
1.8	0.973693	0.973848	0.000155
1.9	0.946350	0.9463	0.00005

TABLE XV (Continued)

Angle(Radian)	Sin(Cordic)	Sin(Correct)	Error
2.0	0.909301	0.909297	0.000004
2.1	0.863220	0.863209	0.000011
2.2	0.808654	0.808496	0.000158
2.3	0.745666	0.745705	0.000039
2.4	0.675476	0.675463	0.000039
2.5	0.598510	0.598472	0.000013
2.6	0.515686	0.515502	0.0000184
2.7	0.427795	0.42738	0.000415
2.8	0.334716	0.334988	0.000272
2.9	0.239074	0.23925	0.000176
3.0	0.140869	0.14112	0.000251
3.1	0.041564	0.0415808	0.0000168
3.2	-0.058654	-0.0583743	0.0002797
3.3	-0.157592	-0.157746	0.000154
3.4	-0.255798	-0.255541	0.000257
3.5	-0.350646	-0.350783	0.000137
3.6	-0.442016	-0.442521	0.000505
3.7	-0.530090	-0.529836	0.000254
3.8	-0.611999	-0.611858	0.000141
3.9	-0.687622	-0.687766	0.000144
4.0	-0.756713	-0.756802	0.000089
4.1	-0.818054	-0.818277	0.000223
4.2	-0.871520	-0.871576	0.000056
4.3	-0.916320	-0.916166	0.000154
4.4	-0.951599	-0.951602	0.000003

TABLE XV (Continued)

Angle(Radian)	Sir(Cordic)	Sin(Correct)	Error
4.5	-0.977539	-0.97753	0.000009
4.6	-0.993774	-0.993691	0.000083
4.7	-0.999877	-0.999923	0.000046
4.8	-0.996032	-0.996165	0.000133
4.9	-0.982422	-0.982453	0.00031
5.0	-0.958801	-0.958924	0.000123
5.1	-0.024901	-0.924815	0.000086
5.2	-0.883422	-0.883455	0.000033
5.3	-0.832153	-0.832267	0.000114
5.4	-0.772583	-0.772765	0.000182
5.5	-0.705505	-0.70554	0.000035
5.6	-0.631530	-0.631267	0.000263
5.7	-0.550659	-0.550686	0.0000027
5.8	-0.464599	-0.464602	0.000003
5.9	-0.373840	-0.373877	0.000037
6.0	-0.279541	-0.279416	0.000125
6.1	-0.182312	-0.182163	0.000149
6.2	-0.082885	-0.0830896	0.0002046

TABLE XVI

THE COMPARISON BETWEEN THE POLYNOMIAL METHOD IMPLEMENTATION RESULT AND THE STANDARD SINE VALUE

Angle(Radian)	Sin(Cordic)	Sin(Correct)	Error
-1.5	-0.997558	-0.997495	0.000063
-1.4	-0.985351	-0.98545	0.000099
-1.3	-0.963378	-0.963558	0.00018
-1.2	-0.932128	-0.932039	0.000089
-1.1	-0.891357	-0.891207	0.00015
-1.0	-0.841552	-0.841471	0.000081
-0.9	-0.783447	-0.783327	0.00012
-0.8	-0.717285	-0.717356	0.000071
-0.7	-0.644287	-0.644218	0.000069
-0.6	-0.564697	-0.564642	0.000055
-0.5	-0.479425	-0.479492	0.000067
-0.4	-0.389404	-0.389418	0.000014
-0.3	-0.295654	-0.29552	0.0001344
-0.2	-0.198730	-0.198669	0.000061
-0.1	-0.099853	-0.0998334	0.0000196
0.0	0.0	0.0	0.000000
0.1	0.099609	0.0998334	0.0001434
0.2	0.198486	0.198669	0.000183
0.3	0.295410	0.29552	0.00011
0.4	0.389160	0.389418	0.000258

TABLE XVI (Continued)

Angle(Radian)	Sin(Cordic)	Sin(Correct)	Error
0.5	0.479248	0.479425	0.000177
0.6	0.564453	0.564652	0.000112
0.7	0.644042	0.644218	0.000176
0.8	0.717041	0.717356	0.000315
0.9	0.783203	0.783327	0.000124
1.0	0.841308	0.841471	0.000163
1.1	0.891113	0.891207	0.0000937
1.2	0.931884	0.932039	0.0001546
1.3	0.963134	0.963558	0.000424
1.4	0.985107	0.98545	0.000343
1.5	0.997314	0.997495	0.000181

consuming task; Task 2 consumes less time; Task 3 consumes still less time; Task 4 consumes the least time of all.

Task 1 and Task 2 are the same algorithm but implemented in different ways, so the sine values will be identical but the execution time may be different. The same applies to Task 3 and Task 4. The programming results in Chapter IV prove this assumption.

Task 1 is performed in an assembly coded program, while Task 2 is performed in a microprogram. According to the description of the microprogramming in Chapter IV, the execution time of Task 2 should be less than that of Task 1. Similarly, Task 4 should have less execution time than Task 3. The programming results in Chapter IV also prove this assumption.

The things that cannot be predicted before going to the computer are whether Task 1 or Task 3 will have less execution time, and whether Task 2 or Task 4 will have less execution time. However, we expect that Task 1 is faster than Task 3 and Task 2 is faster than Task 4. If this is true, it means we can improve the speed of evaluation of trigonometric functions by replacing the conventional polynomial method with the Cordic algorithm. Surprisingly, the programming results in Chapter IV indicate that the conventional polynomial method is faster than the Cordic algorithm for computing trigonometric functions. Although this is disappointing, it is possible to determine exactly how these results were effected.

Although the Cordic algorithm eliminates the necessity of multiplication, some shifting still must be done. In the real Cordic machine, three registers (A,X,Y) can be shifted and added or subtracted simultaneously. When the Cordic algorithm is simulated in this general purpose machine the HP21MX, the shifting and adding or subtracting can only be done sequentially, because the arithmetic unit can only handle one arithmetic operation at a time. In addition to this, the result of shifting and adding must be stored, and then the arithmetic unit for shifting and adding/subtracting of other registers msut be released. After all three registers finish their shifting and adding/subtracting for the current cycle, the next cycle starts. So the shifting and adding/subtracting results of the first register in the previous cycle will be restored, and so on for the second register and thrid. Therefore, when the computer is running, a lot of storing and restoring is being performed, and this is very time-consuming. That is why Task 1 requires more execution time than Task 3. Task 2 implements the Cordic algorithm in a microcode, so it improves the speed of Task 1, but is still slower than Task 3 and Task 4. Task 4 is a microcode, and thus improving the speed of Task 3. Therefore, the conclusions are:

1) The use of the Cordic algorithm for evaluating trigonometric functions without hardware extensions will be slower than using conventional polynomial methods;
2) When using a conventional polynomial method for evaluating the sine function, the microprogram will be two times faster than the assembly coded program;
3) In order to use the Cordic algorithm to improve the speed of evaluation of trigonometric functions, a lot of hardware work must be done in the current HP21MX computer.

With the suprising speed of development of the microprocessor today, it might be very easy to construct a microcomputer which has the features of both the general purpose computer and the Cordic computer in the near future.

A SELECTION BIBLIOGRAPHY

(1) Gear, W. C. Computer Organization And Programming. New York: McGraw-Hill, 1969.

(2) Iverson, K. E. A Programming Language. New York: John Wiley, 1962.

(3) La Lyusternik, O., A. Chervonenkis, and A. R. Yanpol'skii. Handbook for Computing Elementary Functions. New York: Pergamon Press, 1965.

(4) Hayward, J. T. and J. P. Wong, Jr. Approximations For Digital Computers. Princeton, New Jersey: Princeton University Press.

(5) HP21MX Computer Series Reference Manual. Cupertino, California: Hewlett Packard Company, 1974.

(6) HP Microprogramming 21MX Computers Operating and Reference Manual. Cupertino, California: Hewlett Packard Company, 1974.

(7) A Pocket Guide to Hewlett-Packard Computers. Cupertino, California: Hewlett Packard Company, 1974.

(8) A Pocket Guide to Interfacing HP Computers. Cupertino, California: Hewlett Packard Company, 1974.

(9) Volder, J. S. "CORDIC Trigonometric Computation Technique." IRE Transactions on Electronic Computers, EC-8, Sept., 1959, p. 330.

(10) Daggett, D. H. "Decimal-Binary Conversions in CORDIC." IRE Transactions on Electronic Computers, EC-8, Sept., 1959, p. 335.

(11) Meggitt, J. E. "Pseudo-Division and Pseudo-Multiplication Processors." IBM Journal, April, 1962.

(12) Waither, J. S. "A Unified Algorithm for Elementary Functions." Spring Joint Computer Conference, 1971.

(13) Despain, A. M. "Fourier Transform Computers Using CORDIC Iterations." IEEE Transactions on Computers, Vol. c-23, No. 10, Oct., 1974, p. 993.

(14) Cochran, D. S. "Algorithms and Accuracy in the HP 35." Hewlett-Packard Journal, June, 1972.

(15) Schmid, H. and A. Bogacki. "Use of decimal CORDIC for generation of many transcendental functions." Electrical Design News, February, 1973, pp. 64-73.

(16) Richards, R. K. Arithmetic operations in digital computers. New York: Van Nostrand, 1955.

(17) Briggs, H. Logarithmicall arithmetike. London: George Miller, 1631.

APPENDIX A

FUNCTIONAL BLOCK DIAGRAM

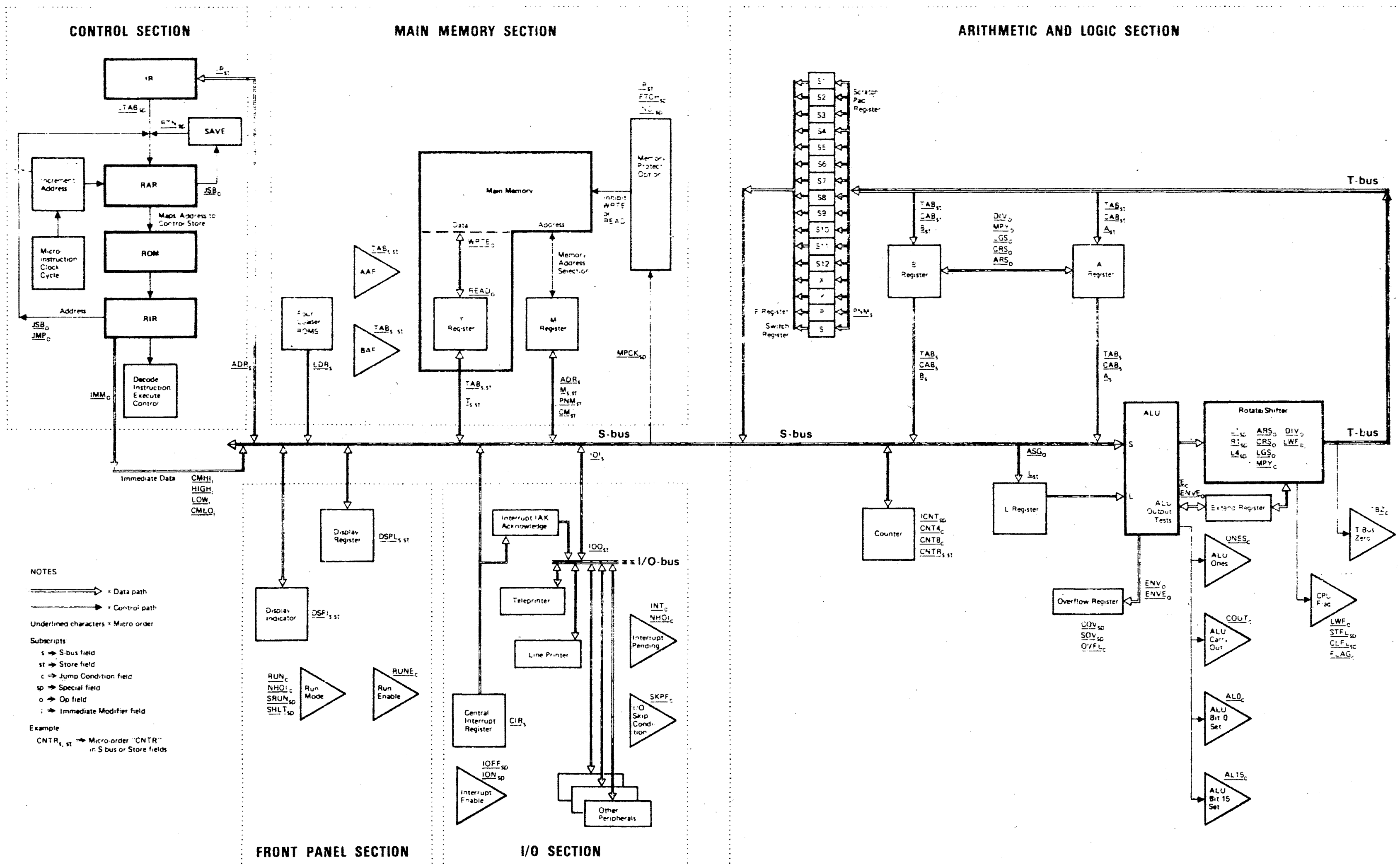

Figure D-1. Functional Block Diagram

APPENDIX B

PROGRAM LISTINGS

```
*******************************************************************************
*  TASK 1 TEST PROGRAM--CORDIC ALGORITHM IMPLEMENTED IN ASSEMBLY             *
*                        CODE PROGRAM                                         *
*  INPUT PARAMETER-- AN ANGLE WHICH MUST BE IN THE RANGE OF                  *
*                        (-360,360) DEGREE                                    *
*  OUTPUT PARAMETER-- SIN VALUE OF THE INPUT ANGLE                           *
*                                                                             *
*******************************************************************************
ASMB,A,B,L,T
*SET UP THE CLOCK INTERRUPT VECTOR ADDRESS
        ORG 14B
        JSB TIME
        ORG 200B
        STF 0B
        CLA
*SET UP INTERRUPT TIME PERIOD TO 0.1 MILISECOND
        OTA 14B
        NOP
*SET REPEATITION COUNT TO 100
START   LDA HD
        STA LCT
        CLA
        CLB
        STA     X
        STA     Y
        STA     COUNT
        FAD     ANG
*CONVERT THE INPUT ANGLE TO THE CORDIC REPRESENTATION
        FDV PI
        RRR 16
        SLA
        JMP AJ
        STA TPS
        AND MC
        SZA
        RRL 1
EN      NOP
        LDA 1B
        JMP ENT
AJ      CAX
        RAR
        IOR FH
        STA SN
        CXA
ST      ASR 1
        ISZ SN
        JMP ST
        JMP EN
ENT     STA RA
ENT1    LDA RA
*INITIATE TIME CLOCK INTERRUPT
*THE CORDIC COMPUTING SEQUENCE STARTS HERE
        STC 14B,C
        CLB
        STB X
        STB Y
        LDB UV
        CLO
        SSA
        JMP CO1
        STB Y
        ADA NRE
B1      LDX SIX
SHT     LBX CON
        STB RV
        STB TES
        LDB Y
```

```
D        NOP
         ISZ TES
         JMP     C1
         JMP     B2
C1       BRS
         JMP D
B2       LDY X
         SSA,RSS
         CMB,INB
         ADY 1B
         LDB X
         STY X
D1       NOP
         ISZ RV
         JMP C2
         JMP B3
C2       BRS
         JMP D1
B3       LDY Y
         SSA
         CMB,INB
         ADY 1B
         STY Y
         LBX A7
         SSA,RSS
         CMB,INB
         ADA 1B
         DSX
         JMP SHT
*EXECUTE THE CORDIC COMPUTING SEQUENCE 100 TIMES FOR THE SAME ANGLE
         ISZ LCT
         JMP ENT1
         CLC 14B
*OUTPUT EXECUTION TIME,SIN AND COS VALUES OF THE INPUT ANGLE
         LDB COUNT
         JSB OUT
         JSB OUT1
         CLB
         STB COUNT
         LDB X
         JSB OUT
         JSB OUT1
         LDB Y
         JSB OUT1
         LDB 0B
         JSB OUT1
         JSB OUT
*INCREASE THE INPUT ANGLE BY 0.1 THEN REPEAT THE PROGRAM
         DLD ANG
         FAD INC
         DST ANG
*SERVICE ROUTINE FOR CLOCK INTERRUPT
         JMP START
TIME     NOP
         STC 14B,C
         ISZ COUNT
         JMP TIME,I
COUNT    OCT 0
CO1      CMB,INB
         STB Y
         ADA RE
         JMP B1
ANG DEC 0.0
PI       DEC 3.14159
FH       OCT 177600
SN       NOP
```

```
UV      OCT 23335
NRE     OCT 140000
RE      OCT 040000
SIX     OCT 16
RV      BSS 1
TES     BSS 1
CON     NOP
        OCT -16
        OCT -15
        OCT -14
        OCT -13
        OCT -12
        OCT -11
        OCT -10
        OCT -7
        OCT -6
        OCT -5
        OCT -4
        OCT -3
        OCT -2
        OCT -1
        NOP
A7      NOP
        OCT 000001
        OCT 000002
        OCT 000005
        OCT 000012
        OCT 000024
        OCT 000050
        OCT 000121
        OCT 000242
        OCT 000505
        OCT 001213
        OCT 002421
        OCT 004773
        OCT 011344
        OCT 020000
        NOP
X       BSS 1
Y       BSS 1
LCT     OCT 177654
HD      OCT 177654
RA      NOP
TPS     NOP
MC      OCT 377
INC     DEC 0.1
```

```
**********************************************************************************************
*  TASK 2 TEST PROGRAM---CORDIC ALGORITHM IMPLEMENTED IN MICROCODE    *
*                         PROGRAM                                       *
*  INPUT PARAMETER-- AN ANGLE WHICH MUST BE IN THE RANGE OF            *
*                    (-360,360) DEGREE                                  *
*  OUTPUT PARAMETER-- SIN VALUE OF THE INPUT ANGLE                     *
*
**********************************************************************************************
ASMB,A,B,L,T
        ORG 14B
*SET UP THE CLOCK INTERRUPT VECTOR ADDRESS
        JSB TIME
        ORG 200B
        STF 0B
        CLA
*SET UP INTERRUPT TIME PERIOD TO 0.1 MILISECOND
        OTA 14B
        NOP
*SET REPEATITION COUNT TO 100
START   LDA HD
        STA LCT
        CLA
        CLB
        STA     X
        STA     Y
        STA     COUNT
        FAD     ANG
*CONVERT THE INPUT ANGLE TO THE CORDIC REPRESENTATION
        FDV PI
        RRR 16
        SLA
        JMP AJ
        STA TPS
        AND MC
        SZA
        RRL 1
EN      NOP
        LDA 1B
        JMP ENT
AJ      CAX
        RAR
        IOR FH
        STA SN
        CXA
ST      ASR 1
        ISZ SN
        JMP ST
        JMP EN
ENT     STA RA
ENT1    LDA RA
*INITIATE TIME CLOCK INTERRUPT
*THE CORDIC COMPUTING SEQUENCE STARTS HERE
        STC 14B,C
        CLB
        CBX
        CBY
        NOP
* THE ENTRY POINT TO THE MICROPROGRAM WHICH PERFORMS THE CORDIC COMPUTING
* SEQUENCE
        OCT 105160
UV      OCT 23335         UNIT VECTOR
*THE ANGLE CONSTANTS
NRE     OCT 140000
ANT1    OCT 020000
ANT2    OCT 011344
ANT3    OCT 004773
```

```
ANT5      OCT 001213
ANT6      OCT 000505
ANT7      OCT 000242
ANT10     OCT 000121
ANT11     OCT 000050
ANT12     OCT 000024
ANT13     OCT 000012
ANT14     OCT 000005
ANT15     OCT 000002
ANT16     OCT 000001
* RETURN TO THIS POINT FROM MICROPROGRAM
          ISZ LCT
          JMP ENT1
          CLC 14B
*OUTPUT EXECUTION TIME, SIN AND COS VALUES OF THE INPUT ANGLE
          LDB COUNT
          JSB OUT
          JSB OUT1
          CLB
          STB COUNT
          LDB X
          JSB OUT
          JSB OUT1
          LDB Y
          JSB OUT1
          LDB 0B
          JSB OUT1
          JSB OUT
*INCREASE THE INPUT ANGLE BY 0.1 THEN REPEAT THE PROGRAM
          DLD ANG
          FAD INC
          DST ANG
*SERVICE ROUTINE FOR CLOCK INTERRUPT
          JMP START
TIME      NOP
          STC 14B,C
          ISZ COUNT
          JMP TIME,I
COUNT     OCT 0
ANG        DEC 0.0
PI        DEC 3.14159
FH        OCT 177600
SN        NOP
          NOP
X         BSS 1
Y         BSS 1
LCT       OCT 177654
HD        OCT 177654
RA        NOP
TPS       NOP
MC        OCT 377
INC       DEC 0.1
```

```
*****************************************************************************
* MICROPROGRAM--USED TO IMPLETMENT THE CORDIC ALGORITHM                      *
*             FOR EVALUATING THE SINE FUNTION                                *
*             THE ANGLE OF THE SINE FUNCTION SHOULD BE STORED IN             *
*             THE REGISTER A BEFORE ENTER THE MICROPROGRAM                   *
*****************************************************************************
$SYMTAB
$ORIGIN=1400
        JMP  NOP  PASS NOP  START
$ORIGIN=1411
START   NOP  NOP  PASS NOP  NOP
*GET THE UNIT VECTOR FROM MAIN MEMORY
        READ NOP  INC  PNM  P
*STORE IT IN REG. X
        NOP  NOP  PASS X    TAB
*GET THE FIRST ANGLE CONTST
        READ NOP  INC  PNM  P
*STORE IT IN REG. S7
        NOP  NOP  PASS S7   TAB
*STORE THE ANGLE OF THE SINE FUNCTION IN REG. S6
        NOP  NOP  PASS S6   A
*IF THE ANGLE IS LESS THAN 180 DEGREE,BRANCH  TO EN1
        JMP  CNDX AL15 RJS  EN1
*GET THE TWO'S COMPLEMENT OF S7
        NOP  NOP  CMPS S7   S7
        NOP  NOP  INC  S7   S7
*GET TWO'S COMPLENT OF X
        NOP  NOP  CMPS X    X
        NOP  NOP  INC  X    X
*STORE X IN Y
EN1     NOP  NOP  PASS Y    X
*CLEAR X REG.
        NOP  NOP  ZERO X    NOP
*S6=S6+S7
        NOP  NOP  PASS L    S7
        NOP  NOP  ADD  S6   S6
*S4=-14
        IMM  NOP  LOW  S4   362B
*CLEAR S3
        IMM  NOP  LOW  S3   0B
        NOP  NOP  PASS S5   X
        NOP  NOP  PASS L    Y
        JMP  NOP  NOP  NOP  BK1
*INITIALIZE THE COUNTER
BK      NOP  NOP  PASS CNTR S3
*RIGHT SHIFT B REG. BY THE NUMBER IN THE COUNTER,
*THEN STORE THE SHIFTING RESULT IN S5
        NOP  RPT  PASS B    X
        ARS  R1   PASS B    B
        NOP  NOP  PASS S5   B
*SET COUNTER AGAIN
        NOP  NOP  PASS CNTR S3
        NOP  RPT  PASS B    Y
        ARS  R1   PASS B    B
        NOP  NOP  PASS L    B
*GET NEXT ANGLE CONSTANT
BK1     READ NOP  INC  PNM  P
        NOP  NOP  PASS S7   TAB
        NOP  NOP  PASS S6   S6
*TEST THE ANGLE ,IF GREATER THAN 180 DEGREE GO TO EN2
        JMP  CNDX AL15 NOP  EN2
        NOP  NOP  SUB  X    X
        NOP  NOP  PASS L    S5
        NOP  NOP  ADD  Y    Y
        NOP  NOP  PASS L    S7
        NOP  NOP  SUB  S6   S6
        JMP                 JN
EN2     NOP  NOP  ADD  X    X
        NOP  NOP  PASS L    S5
        NOP  NOP  SUB  Y    Y
        NOP  NOP  PASS L    S7
        NOP  NOP  ADD  S6   S6
JN      NOP  NOP  INC  S4   S4
        JMP  CNDX TBZ  NOP  EXIT
        NOP  NOP  DEC  S3   S3
        JMP                 BK
EXIT    NOP  RTN  PASS NOP  NOP
$END
```

```
*************************************************************************
*  TASK 3 TEST PROGRAM--POLYNOMIAL METHOD IMPLEMENTED IN ASSEMBLY      *
*                         PROGRAM                                       *
*  INPUT PARAMETER--AN ANGLE RANGED FROM -90 DEGREE TO 90 DEGREE       *
*  OUTPUT PARAMETER--THE SINE VALUE OF THE INPUT ANGLE                  *
*************************************************************************
ASMB,A,B,L,T
*SET UP THE CLOCK INTERRUPT VECTOR ADDRESS
        ORG 14B
        JSB TIME
        ORG 200B
        STF 0B
        CLA
        OTA 14B
        NOP
START   LDA HD
        STA LCT
        CLA
        CLB
        CAX
        CAY
        STA COUNT
        STC     14B,C
ENT1    LDA ANG
        MPY ANG
        ASL 1
        STB SQ
        LDA     1B
        CLB
        MPY C5
        ADB C3
        LDA     1B
        CLB
        MPY SQ
        ASL 3
        ADB C1
        LDA 1B
        CLB
        MPY ANG
        STB ANG
        STB ANS
        ISZ LCT
        JMP ENT1
        CLC 14B
*OUTPUT EXECUTION TIME,SIN AND COS VALUES OF THE INPUT ANGLE
        LDB COUNT
        JSB OUT
        JSB OUT1
        CLB
        STB COUNT
        LDB ANS
        JSB OUT
        JSB OUT1
        LDB ANG
        JSB OUT1
*INCREASE THE INPUT ANGLE BY 0.1 THEN REPEAT THE PROGRAM
        LDA INC
        ARS,ARS
        ARS,ARS
        ARS
        ADA ANG
        STA ANG
        JMP START
*SERVICE ROUTINE FOR CLOCK INTERRUPT
TIME    NOP
        STC 14B,C
        ISZ COUNT
        JMP TIME,I
COUNT   OCT 0
ANG     OCT 0
LCT     OCT 177654
HD      OCT 177654
ANS     OCT 0
SQ      OCT 0
C1      DEC 0.999892
C3      DEC -0.1659685
C5      DEC 0.0076031915
INC     DEC 0.1
```

```
*****************************************************************************
*  TASK 4 TEST PROGRAM--POLYNOMIAL METHOD IMPLEMENTED IN MICROPROGRAM*
*                          PROGRAM                                          *
*  INPUT PARAMETER--AN ANGLE RANGED FROM -90 DEGREE TO 90 DEGREE         *
*  OUTPUT PARAMETER--THE SINE VALUE OF THE INPUT ANGLE                    *
*****************************************************************************
ASMB,A,B,L,T
*SET UP THE CLOCK INTERRUPT VECTOR ADDRESS
        ORG 14B
        JSB TIME
        ORG 200B
        STF 0B
        CLA
        OTA 14B
        NOP
START   LDA HD
        STA LCT
        CLA
        CLB
        CAX
        CAY
        STA COUNT
        STC     14B,C
*ENTRY POINT OF THE MICROGRAM
ENT1    OCT 105160
ANG     OCT 0
C1      OCT 77774
C3      OCT 125406
C5      OCT 76222
ANS     OCT 0
*THE MICROPROGRAM RETURNS THE CONTROL TO THIS POINT
        ISZ LCT
        JMP ENT1
        CLC 14B
*OUTPUT EXECUTION TIME,SIN AND COS VALUES OF THE INPUT ANGLE
        LDB COUNT
        JSB OUT
        JSB OUT1
        CLB
        STB COUNT
        LDB ANS
        JSB OUT
        JSB OUT1
        LDB ANG
        JSB OUT1
*INCREASE THE INPUT ANGLE BY 0.1 THEN REPEAT THE PROGRAM
        LDA INC
        ARS,ARS
        ARS,ARS
        ARS
        ADA ANG
        STA ANG
        JMP START
*SERVICE ROUTINE FOR CLOCK INTERRUPT
TIME    NOP
        STC 14B,C
        ISZ COUNT
        JMP TIME,I
COUNT   OCT 0
LCT     OCT 177654
HD      OCT 177654
SQ      OCT 0
INC     DEC 0.1
        END
```

```
****************************************************************************************
*  MICROPROGRAM---USED TO EVALUATE SIN FUNCTION BY IMPLEMENTING THE                    *
*                     POLYNOMIAL METHOD                                                *
****************************************************************************************
$SYMTAB
$ORIGIN=1400
         READ NOP  INC  PNM  P
*STORE THE VALUE X IN S2 AND S9
         NOP  NOP  PASS S2   TAB
         NOP  NOP  PASS S9   S2
*STORE THE VALUE C1 IN S1
         READ NOP  INC  PNM  P
         NOP  NOP  PASS S1   TAB
*STORE THE VALUE C3 IN S3
         READ NOP  INC  PNM  P
         NOP  NOP  PASS S3   TAB
*STORE THE VALUE C5 IN S5
         READ NOP  INC  PNM  P
         NOP  NOP  PASS S5   TAB
*COMPUTE X*X
         NOP  NOP  PASS A    S2
         JSB  NOP  NOP  NOP  MPY
         ARS  L1   PASS B    B
*STORE X*X IN S6
         NOP  NOP  PASS S6   B
*COMPUTE C5*X*X
         NOP  NOP  PASS A    B
         NOP  NOP  PASS S2   S5
         JSB  NOP  NOP  NOP  MPY
         NOP  NOP  PASS L    S3
*COMPUTE C3+C5*(X*X)
         NOP  NOP  ADD  A    B
*COMPUTE (X*X)*(C3+(C5(X*X)))
         NOP  NOP  PASS S2   S6
         JSB  NOP  NOP  NOP  MPY
*ADJUST THE SCALE FACTOR
         ARS  L1   PASS B    B
         ARS  L1   PASS B    B
         ARS  L1   PASS B    B
*COMPUTE C1+(X*X)*(C3+C5*(X*X))

         NOP  NOP  PASS L    S1
         NOP  NOP  ADD  A    B
*COMPUTE X(C1+X*X*(C3+C5*(X*X)))
         NOP  NOP  PASS S2   S9
         JSB  NOP  NOP  NOP  MPY
*SAVE THE RESULT IN MAIN MEMORY
         NOP  NOP  PASS T    B
         WRTE NOP  INC  PNM  P
*RETURN TO MACROPROGRAM
RETURN   NOP  RTN  PASS NOP  NOP
*SUBROUTINE FOR COMPUTING THE MULTIPLICATION OF TWO INTEGERS
MPY      NOP  COV  PASS S7   A
         NOP  NOP  ZERO B    NOP
         NOP  NOP  PASS L    S2
         NOP  RPT  PASS CNTR B
         MPY  R1   ADD  B    B
         NOP  NOP  PASS NOP  S7
         JMP  CNDX AL15 RJS  *+2
         NOP  NOP  SUB  B    B
         NOP  NOP  PASS NOP  S2
         JMP  CNDX AL15 RJS  RETURN
         NOP  NOP  PASS L    S7
         NOP  RTN  SUB  B    B
$END
```

```
*****************************************************************************************
* PRINTING ROUTINE---USED TO PRINT THE CONTENT OF B-REGISTER                            *
*****************************************************************************************
OUT1    NOP
        DST SAV
        STX SAV1
        LDA TTY
        OTA 11B
        LDA BL
        OTA 11B
        STC 11B,C
        SFS 11B
        JMP *-1
        LDA SIXT
        CAX
        CLA
LOP     RRL 1
        ADA ASC
        OTA 11B
        STC 11B,C
        CLA
        SFS 11B
        JMP *-1
        ISX
        JMP LOP
        DLD SAV
        LDX SAV1
        JMP OUT1,I
        END
```

```
******************************************************************************
* CARRIAGE CONTROL ROUTINE--RETURN THE CARRIAGE TO THE BEGINING OF*
                              THE LINE AND FEED ONE LINE            *
******************************************************************************
OUT     NOP
        DST SAV
        CLC 0,C
        LDA TTY
        OTA 11B
        LDA CR
        OTA 11B
        STC 11B,C
        SFS 11B
        JMP *-1
        LDA LF
        OTA 11B
        STC 11B,C
        SFS 11B
        JMP *-1
        DLD SAV
        JMP OUT,I
ASC     OCT 60
TTY     OCT 120000
SIXT    OCT 177760
CR      OCT 215
LF      OCT 12
SAV     NOP
        NOP
SAV1    NOP
BL      OCT 240
        END
```

VITA 2

Peihsung Thomas Hu

Candidate for the Degree of

Master of Science

Thesis: THE CORDIC ALGORITHM IMPLEMENTATION FOR TRIGONOMETRIC FUNCTION EVALUATION IN HP21MX

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Taipei, Taiwan, Republic of China, July 2, 1950, the son of Mr. and Mrs. B. Y. Hu.

Education: Graduated from Chengko High School, Taipei, Taiwan, Republic of China, in June, 1968; received Bachelor of Science degree in Electrical Engineering from Chiao Tung University, HsinoHu, Taiwan, Republic of China, in June,1972; completed requirements for the Master of Science degree at Oklahoma State University in May, 1978.

Professional Experience: Graduate teaching assistant, Department of Computing and Information Sciences, Oklahoma State University, 1975-1976; Software specialist, Atkins & Merril Training Equipment Company, 1976-present.