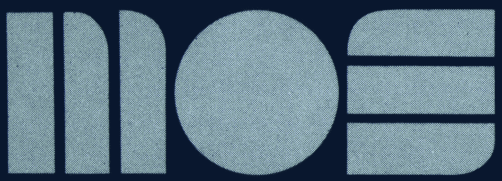


INSTRUTEK
8700 Horsens · Tlf. 05 · 61 11 00

MICROCOMPUTERS

MICROCOMPUTERS

MICROCOMPUTERS

MICROCOMPUTERS

KIMath
SUBROUTINES PROGRAMMING MANUAL

MICROCOMPUTER FAMILY
KIM MATHEMATICS SUBROUTINES
Programming Manual

Copyright by MCDS
3. englische Auflage 1977
Auflagenhöhe 1. bis 3. : 15.000

The information in this manual has been reviewed and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. The material in this manual is for informational purposes only and is subject to change without notice.

MCDS verfügt über alle Rechte der deutschen und englischen Ausgabe. Nachdrucke und Vervielfältigungen, auch auszugsweise, sind nur mit ausdrücklicher Genehmigung der MCDS gestattet.

Auch bei dieser 3. englischen Auflage freuen wir uns über kritische Anregungen des Leserkreises, die zur kontinuierlichen Verbesserung der Qualität weiterer Auflagen Verwendung finden.

Satz, Druck und Vertrieb von MCDS Microcomputer Datensyteme GmbH, Luisenplatz 4, Postfach 110868, D-6100 Darmstadt, Tel. 06151/20302, Telex 04 19 390 hysy d

MOS TECHNOLOGY · Frankfurter Str. 171—175 · D—6378 Neu-Isenburg
Telefon (06102) 8003 · Telex 04 185663

TABLE OF CONTENTS

LIST OF APPENDICES

INTRODUCTION

This manual contains instructions on the use of the KIMath subroutine package. KIMath provides the software for doing floating-point (decimal) arithmetic on the KIM microcomputer system.

Because of the wide availability of calculators and larger computers we take decimal arithmetic for granted. Perhaps you were surprised when you first read the 6502 microcomputer programming manual and discovered that the 6502 instruction set could do addition and subtraction of whole numbers only in the range of +128 to -127! If you tried writing a program to multiply two eight-bit binary words to produce a sixteen-bit result, you will appreciate the ability in KIMath to multiply two sixteen-digit BCD numbers just by calling a subroutine.

KIMath gives you the capability to add, subtract, multiply, divide and calculate square roots. In addition, KIMath can calculate logs and antilogs as well as tangents and arc tangents. Other common mathematical functions can be calculated by combining these basic operations. KIMath also has special subroutines to make it easy to evaluate polynomial expressions, which can be used to approximate most mathematical functions.

Since your KIM system, like any microcomputer, has to break large numbers into several successive words in memory and then work on them one word at a time, floating-point arithmetic is much more time- and memory-consuming than simple binary arithmetic. A single floating-point number may use as many as 18 words of memory in your KIM system, and the multiplication of two 16-digit numbers may take as long as 40 milliseconds--40,000 machine cycles! The same multiplication using only 4 digits of precision would take only 10 milliseconds, so if you will accept less precision you can speed up your calculations.

To make this possible KIMath allows you to specify the number of digits to be used in any calculation. You may use the same precision for your entire program or you may adjust the precision depending on the needs of each step in your calculations.

The KIMath subroutines are an ideal addition to the KIM resident assembler. Naturally, the assembler is not required in order to use KIMath.

Because each floating-point number is stored in several memory words, KIMath provide utility routines to move blocks of words corresponding to each number to and from the locations required to perform the calculations.

Finally, KIMath allows floating point values to be stored in several different representations or formats to improve computational efficiency while conserving memory. Routines are provided to convert between these formats.

Chapter 1 of this manual gives you an overview of the concepts used in KIMath. Chapter 2 shows how KIMath could be used to solve a simple equation. Chapter 3 examines the different storage formats and Chapters 4 and 5 cover the use of the individual subroutines.

Several appendices are provided for the user who desires a more rigorous explanation of the techniques used in designing KIMath. A storage map and a complete program listing are also included.

The KIMath subroutines were carefully designed and thoroughly tested to insure accuracy and correct operation. The final decision of their suitability, however, rests with the user and no warranty of fitness is implied by this document. If you suspect an error in these routines, contact the Manager of Product Support at MOS Technology Sales, Inc. 901 California Avenue, Palo Alto, California 94304.

This manual assumes familiarity with 650X assembly language. You should first study the KIM Resident Assembler if necessary.

CHAPTER 1

Overview

1.0 Basic Concepts

KIMath is a package of subroutines designed for use with any MCS650X microprocessor-based system. The subroutines assist the user in doing the floating-point arithmetic necessary for many business and scientific applications. In particular, KIMath is designed to work with the KIM resident assembler, although KIMath may be integrated into any MCS650X program.

1.1 Definitions

Before examining the operation of KIMath, here are definitions of some terms used throughout this manual:

WORD or BYTE - the basic unit of data used in the KIM system. It is composed of eight BITS or binary digits, each of which may only have values of Ø or 1. The microprocessor can only operate on one word at a time.

REGISTER - has a special meaning in this manual, where we define it to mean a group of adjacent words in memory. Since the numbers manipulated by KIMath are too big to store in a single word, we call the group of words used to hold a single floating-point number a register. This usage should not be confused with the registers A, X, Y, stack pointer and status which are single words stored within the microprocessor, not the memory.

POINTER - two adjacent words in memory containing the address of another memory location. The programming convention in the 6502 is to store the two low-order hex characters of the address in the first word and the two high-order characters in the second word. For example, given:

Label	Location	Contents
POINTL	17FA	00
POINTH	17FB	1C

we could say that the register (POINTL, POINTH) is a pointer which currently points to the address 1CØØ.

ROUTINE - a segment of machine-language code which performs a specified operation. In this document it is synonymous with a subroutine.

ARGUMENT - the KIMath routines expect to find the numbers they are to operate on in fixed locations in memory. When the KIMath routine ADD is called by a program, it will take data from two fixed locations, add them together and put the result in a third fixed location. The data in the input locations are the arguments for the ADD routine. It is usually necessary to transfer the data from other registers in memory into the argument registers and transfer the calculated result to another memory register for storage. KIMath provides routines for moving registers about in memory.

WHOLE NUMBER or INTEGER - a number which does not contain a decimal point.

FLOATING-POINT NUMBER - a number containing a decimal point. Note: 12 is an integer while 12.0 is a floating-point number.

1.2 Number Systems

All modern digital computers are based on the binary system, where numbers are represented as groups of signals which are on or off, 1 or Ø. For instance, the decimal number 39 is represented in binary as 100111. For convenience, binary numbers are often converted to groups of four and represented in base 16 or hexadecimal. Decimal 39 is represented as:

2	7	hexadecimal
0010	0111	binary

Because our daily arithmetic is done in decimal, the hexadecimal format is still difficult to interpret. The MCS6502 microprocessor also has the capability to do arithmetic in a modified binary format known as BCD or binary coded decimal. In this system each eight-bit word is divided into two four-bit fields, each representing a decimal digit. Decimal 39 is now represented as:

3	9	decimal
0011	1001	BCD

memory words

Although less efficient in use of storage, this representation is more readily understood. Appendix H of the 6502 Programming Manual contains a review of binary and BCD arithmetic.

We can represent any whole number as a series of BCD digits, with two digits packed in each memory word. The next problem arises when we wish to store numbers such as 3.14159 or 1276.3333. These numbers contain decimal fractions and BCD notation contains no provisions for decimal points. Very large or very small numbers such as 987000000000 or 0.000000625 often have large numbers of leading or trailing zeros which expand storage requirements. To overcome these problems, KIMath uses "scientific notation" which stores decimal numbers in two parts. The number is first reduced to a single whole number followed by a decimal fraction. This is called the mantissa. The second part is the number of decimal places we must shift the new decimal point to get back to the original number. This is called the exponent (scientific notation is also referred to as exponential notation). If the decimal point must be shifted to the right the exponent is a positive number; if a left shift is necessary the exponent is negative. For instance:

987000000000 becomes 9.87 E+11

0.000000625 becomes 6.25 E-7

(An E for exponent is commonly used to separate the two values.) Similarly:

-6.273411 E+3 is -6273.411

which is an example of a number with a negative mantissa and a positive exponent.

1.3 How Will I Use KIMath in My Program?

Your program will use data of two types: variables whose values will change from one program run to the next or within a single run, and constants which will always have the same value. For example, a simple program to compute the area of a circle given its radius will solve the equation:

$$AREA = 3.14\ R^2$$

The values for AREA and R will change each time you run the program; they are variables. The values 3.14 (pi) and 2 (the power to which R is raised) are constants.

When you write your program in assembly language you define memory locations to store this data. When you reserve storage for data used with KIMath, you will have to reserve several words of storage for each variable and constant and you will have to define a register for each data value. The size of each register will depend on the number of digits you wish to store--that is, how much precision you wish to maintain in your calculations.

When you wish to perform calculations using KIMath, you will have to transfer the value of the variables or constants from their memory storage registers into the argument registers (RX and RY) of KIMath, call the appropriate KIMath routine to do the calculation, and then transfer the answer or result from the KIMath result register (RØ) back to a variable storage register. KIMath provides routines for doing these register moves.

In addition, KIMath provides for a variety of data formats to minimize storage requirements and speed computation. KIMath provides routines for converting data between these different formats.

1.4 Summary of KIMath Calculation Subroutines

Subroutine	Action	Description
ADD	RX + RY → RØ	This routine allows the user to add two floating-point numbers. The user may select the number of digits in the mantissas of the arguments.
SUB	RX - RY → RØ	This routine allows the user to subtract two floating-point numbers. The user may select the number of digits in the mantissas of the arguments.
MULPLY	RX * RY → RØ	This routine allows the user to multiply two floating-point numbers. The user may select the number of digits in the mantissas of the arguments.
DIVIDE	RX ÷ RY → RØ	This routine allows the user to divide two floating-point numbers. The user may select the number of digits in the mantissas of the arguments.

1.4 Summary of KIMath Calculation Subroutines (Continued)

Subroutine	Action	Argument Range	Description
SQRT	$\sqrt{RX} \rightarrow RB$	{1, 100}	This routine allows the user to form the square root of a floating-point number. The user may select the number of digits in the mantissa of the argument.
LOG	$LOG_{10}(RX) \rightarrow RB$	$\left\{\frac{1}{\sqrt{10}}, \sqrt{10}\right\}$	This routine allows the user to form the common logarithm of a floating-point number. The user may select the number of digits in the mantissa of the argument, but at most 14 will be used and at most 8 will be returned.
TENX	$10^{RX} \rightarrow RB$	{0, 1}	This routine allows the user to form the common antilog of a floating-point number. The user may select the number of digits in the mantissa of the argument, but at most 12 will be used and at most 8 will be returned.
TANX	$TAN(RX) \rightarrow RB$	{0, 1}	This routine allows the user to form the tangent of a floating-point number. The user may select the number of digits in the mantissa of the argument, but at most 14 will be used and at most 8 will be returned. The argument and result are in radians.
ATANX	$ARCTAN(RX) \rightarrow RB$	{0, 1}	This routine allows the user to form the arc tangent of a floating-point number. The user may select the number of digits in the mantissa, but at most 14 will be used and at most 8 will be returned. The argument and result are in radians.

1.5 KIMath Formats

Several variations on exponential notation are used in KIMath.

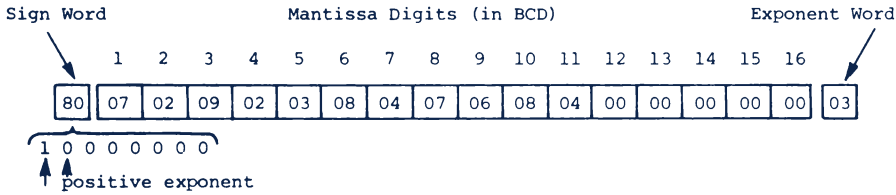
1.5.1 Computational Format

Before actual computation can take place, every number transferred to a KIMath argument register must be converted to an eighteen-word format:

a. The first word contains the sign of the mantissa in bit 7 (1 if minus, 0 if plus) and the sign of the exponent in bit 6.

b. The next sixteen words contain the sixteen digits of the mantissa, one digit per word, using BCD notation.

c. The last word contains the value of the exponent (which must be between 0 and 99) as two BCD numbers. For example: a value of -7292.3847684 would be stored in computational format as:



This format allows KIMath to calculate quickly since every floating-point number is contained in the same number of words and each word contains only a single digit (except for the exponent word).

This is a very inefficient use of storage since the left half of each word in the mantissa always contains a zero, and each mantissa contains sixteen digits even if many are just trailing zeros.

1.5.2 Packed Format

To avoid this waste of storage KIMath provides a more efficient format for storing data when it is not being used in a computation:

a. The first word contains the signs of the mantissa and exponent as in computational format.

b. Succeeding words each contain two digits of the mantissa in BCD. The number of words used is defined by the user.

c. The final word contains the two digits of the exponent, as in computational format.

If the user had decided to use 12 digits of precision in storage, the value of -7292.3847684 would be stored as:

thus using eight words of storage instead of eighteen. KIMath contains routines to convert between computational and packed format. Packed format is the form

usually used for memory registers storing program variables. It is your responsibility to keep track of how many digits of precision are used for each variable, since KIMath will need that information when moving the memory register to the KIMath argument registers and converting to computational format. You may choose to use the same precision for all storage registers to simplify this task.

1.5.3 Unpacked (ASCII) Format

The ASCII code used by the KIM-1 serial communications interface represents each character typed in or out as two hexadecimal characters packed in a single word: the character 'A' is 41, a blank is 2Ø, a carriage return is ØD. The digits Ø through 9 are coded as themselves plus 30: '4' is 34, '7' is 37, etc.

To make interfacing easier, KIMath has an unpacked format which uses ASCII codes rather than BCD.

a. The first word contains the sign bits for the mantissa and exponent as used for computational format.

b. Up to 16 words (depending on the precision desired) are used for the mantissa. Each word contains one digit of the mantissa in ASCII code.

c. The last two words contain the two digits of the exponent, also in ASCII.

Using six digits of precision the value

-0.003764 (or -3.764 E-3)

would be stored as:

This format is usually used only when preparing to transfer data to or from a line of terminal input or output.

1.5.4 Constant Format

KIMath calculates functions by solving polynomial equations which closely approximate the desired function. Because the polynomials have many terms and the coefficients have differing precision requirements, a special format is used in KIMath to store constants. The format allows all constants to

be stored together in a list without having to specify elsewhere the precision of each constant. The constant format is composed of:

a. A sign byte, as in the other formats.

b. Up to eight bytes in packed BCD format (two BCD characters per byte) containing the mantissa.

c. An exponent byte of two characters. The first (leftmost) character is always a hexadecimal 'F' and the second character is the one-digit BCD exponent. Thus, constants may only have exponents between -9 and +9.

Thus the constant 32767.413 (3.2761413 E+4) will be stored as:

Thus, KIMath can read a constant from memory by just knowing its starting location. It will continue to scan successive bytes until it locates one containing an 'F' and know it has reached the end of the constant. The next byte will contain the sign byte of the next constant.

The KIMath routine USRLKP is used to transfer a constant format value from memory to the argument register RY and convert it to computational format.

1.6 Summary of Formats

KIMath supports four different formats:

a. Computational Format - used for the argument and result registers RX, RY and RZ. Although inefficient in use of storage, they permit quick calculation.

b. Packed BCD Format - most efficient in use of storage.

c. Unpacked (ASCII) Format - less efficient in use of storage, but easier to convert to and from external devices.

d. Constant Format - efficient in use of storage and a good way to store tables, but limited in the size of the exponents which can be used.

CHAPTER 2

A Sample Program Using KIMath

2.0 Throughout the remainder of this manual we will develop the techniques required to create the following program:

An engineer wishes to create a program to find the resonant frequency of a tuned circuit, given values for L, the inductance, and C, the capacitance, in the circuit. The formula is:

$$f = \frac{1}{2\pi \sqrt{LC}}$$

The program will have to contain the following operations:

1. The starting address of the program must be defined and storage registers for L, C, and the constants 2π and 1 must be created.
2. The precision of the calculations must be defined to KIMath.
3. The value of L must be read in and transferred to a KIMath argument register.
4. The value of C must be read in and transferred to a KIMath argument register.
5. L must be multiplied by C.
6. Move the answer to step 5 to a KIMath argument register. Compute the square root. Since the square root routine in KIMath will only accept numbers between 1 and 100 we must adjust the exponent of the result of step 5.
7. Get the first constant (2π) and transfer it to an argument register.
8. Move the computed square root to another argument register and multiply.
9. Get the second constant (1) and put in an argument register.
10. Move the result of step 8 to an argument register and do the division.
11. Move the result of the calculation back to the register originally containing L and type it out.

To summarize as an assembly language program,

```
;(1)   define storage
;(2)   define precision
```

```
;(3)    read L and transfer to RX
;(4)    read C and transfer to RY
;(5)    compute L times C--answer stored in RZ
;(6)    move RZ to RX and
;       compute square root--adjust exponent of result
;(7)    get 2π to RX
;(8)    move RZ to RY and multiply
;(9)    get 1 (constant) and put in RX
;(10)   move RZ to RY and divide
;(11)   move RZ back to L and print out
.END
```

To start with, our program only contains comments. As we learn more about KIMath we will begin to write the assembly language statements to implement the program.

To help you visualize what is happening in the program, the following table shows the contents of the two argument registers (RX, RY) and the result register (RZ) after each program step:

At the end of step #	RX	RY	RZ
1	--	--	--
2	--	--	--
3	L	--	--
4	L	C	--
5	L	C	LC (L times C)
6	LC	C	$\sqrt{LC}$ (adjusted)
7	2π	2π*	$\sqrt{LC}$
8	2π	$\sqrt{LC}$	$2\pi\sqrt{LC}$
9	1	1*	$2\pi\sqrt{LC}$
10	1	$2\pi\sqrt{LC}$	$\frac{1}{2\pi\sqrt{LC}}$

*The constant appears in both RX and RY because it is moved from storage to RY and then to RX.

CHAPTER 3

Defining Storage for KIMath Variables and Constants

3.0 In the last chapter you learned that KIMath uses several different formats for memory storage registers used in KIMath calculation. Now let us see how we set aside memory to hold these variables and constants.

3.1 Defining Precision

If we define an even number NDIG which is the Number of DIGits of precision we wish to use, storing a number of that precision will take:

a. NDIG + 3 words of storage in Unpacked format.

b. NDIG/2 + 2 words of storage in Packed format.

c. NDIG/2 + 2 words of storage in Constant format. (Constants may be stored in fewer digits if they have leading or trailing zeros.)

d. Computational format always takes 18 words.

In our sample program (see Chapter 2) we defined two variables, L and C. Since we are reading them in from a terminal we will store them in Unpacked (ASCII) format. We also needed two constants (2π and 1) and will store them in constant format.

Since we may wish to change the precision of our program, we will define the variable NDIG to the assembler and use it to calculate our storage. The following code will define storage for our sample program:

```
;(1)   define storage
NDIG = 8; we will carry 8 digits of precision
* = $5000; start the program at location 5000 (hex)
L    * = * + NDIG + 3; reserve storage for L
C    * = * +NDIG +3; reserve storage for C
TWOPI .BYTE $00, $62, $83, $18, $54, $F0; define 2π
ONE   .BYTE $00, $10, $F0; define 1.0
```

Note that constant format allows us to eliminate the trailing zeros in the constant ONE. The symbol "*" stands for the current value of the program counter. Refer to the assembler documentation if you are not familiar with assembler directives.

Our program storage map will now be:

5000 - 500A	storage for L
500B - 5015	storage for C
5016 - 501B	storage for 2π constant
501C - 501E	storage for 1.0 constant
501E -	program storage

If we later re-assemble our program and change the value of NDIG to 4 or 10, to increase or decrease the precision of our calculations, the amount of storage allocated will automatically decrease or increase.

CHAPTER 4

Introduction to the KIMath Subroutines

4.0 Now that we have examined the data formats and storage requirements of KIMath we are ready to examine the subroutines themselves. The following table summarizes the KIMath subroutines, their arguments and their functions:

Summary of KIMath Subroutines

Subroutine	Arguments	Function
SAVXY	None	Save processor X and Y index registers.
RCLXY	None	Recall previously saved X and Y registers.
IPREC	PREC, EXTRA	PREC + EXTRA → N
PLOADX PLOADY	ARGXL, ARGXH ARGYL, ARGYH PREC	Move user register in packed format to RX or RY and change to computational format.
ULOADX ULOADY	ARGXL, ARGXH ARGYL, ARGYH PREC	Move user register in unpacked format to RX or RY and change to computational format.
PSTRES USTRES	RESL, RESH PREC	Move contents of RB to user register in packed or unpacked format.
USRLKP	KONL, KONH, NKON	Transfer constant data to RY and convert to computational format.
POLY	KONL, KONH, NKON, DEG	Evaluates a polynomial.
CLRX CLRY CLRB	None	Clear RX, RY, or RB
DECHEX	CNT	Converts contents of CNT from decimal to hexadecimal notation.
XSY	RX, RY	Exchange contents of RX and RY.

4.1 Working Storage

The working storage area used by the KIMath subroutines is divided into three parts: the computational registers (RA, RB, RQ, RM and RN) which are not available to the user; the argument/result registers (RX, RY, RB); and the pointers, counters and indicators in page zero.

4.1.1 The Argument/Result Registers - RX, RY, RØ

The registers RX and RY are the argument registers and RØ is the result register. This means that when RX and RY are set to specific values and, for example, the MLTPLY routine is called, then the product is returned in RØ. These registers are each 18 bytes long to permit a maximum of 16 digits for the mantissa. The registers RX, RY and RØ are CLeaRed directly by means of the subroutines CLRX, CLRY and CLRØ, respectively. The registers RX, RY, RØ contain a sign byte (SX, SY, SØ) and an exponent byte (EX, EY, EØ). The sign byte is the first and the exponent byte the last. Remember that data in RX, RY or RØ must be in computational format.

These registers may be transferred about by means of the subroutines whose names are of the form MVsd, where "s" stands for source and "d" for destination. For example, MVXY MoVes the contents of RX to RY.

4.1.2 The Pointers, Counters and Indicators

Before calling any KIMath routine, you may wish to SAVe the processor's X and Y registers in locations TMPX and TMPY by means of the routine SAVXY. The X and Y registers are ReCaLled by calling RCLXY.

Before calling an arithmetic or transcendental KIMath routine you must specify the number of digits of PRECision desired by loading locations PREC and EXTRA with two unsigned, binary values. The first value, PREC, specifies the number of digits of mantissa which the user expects in the RØ register after completion of a calculation and PREC+EXTRA specifies the number of digits of precision used by the subroutine in computing the result. The limitations on these values are: $0 < PREC+EXTRA \leq 16$ and:

Function	Internal Precision*	Precision* Returned
ADD	PREC+EXTRA	PREC
SUB	PREC+EXTRA	PREC
MLTPLY	PREC+EXTRA	PREC
DIVIDE	PREC+EXTRA	PREC
LOG	14	Lesser of PREC or 8
TENX	12	Lesser of PREC or 8
TANX	14	Lesser of PREC or 8
ATANX	14	Lesser of PREC or 8
SQRT	PREC+EXTRA	PREC

*Note: Here the word "precision" refers not to error but to the number of digits used in calculation.

The sum PREC+EXTRA may be computed by calling the routine named IPREC. This routine computes the sum and stores the result in N. N is the byte which the arithmetic and transcendental routines use to decide the precision of internal calculation. If the user so desires, the use of PREC and EXTRA may be avoided and N used directly. However, the routines LOG, TENX, TANX and ATANX may change any value placed in N.

The bytes in page Ø named ARGXL, ARGXH, ARGYL, and ARGYH are used to contain the addresses of user-defined registers in the RAM memory area. By using these address pointers and the subroutines PLOADX, PLOADY, ULOADX and ULOADY, the user loads the KIMath argument registers from any place in memory.

The bytes RESL and RESH are used in conjunction with the KIMath routines PSTRES (Packed SToring of RESults) and USTRES (Unpacked SToring of RESults) to place the contents of RØ in a user-defined register, which is specified by the pointer (RESL, RESH).

The bytes KONL, KONH are used together with DEG and KIMath routines USRLKP and POLY to evaluate any polynomial, subject, of course, to the constraints that the coefficients have fewer than 16 digits of mantissa and that the exponent has only one digit. More shall be said about these techniques in the section on Advanced Applications.

CHAPTER 5

The KIMath Subroutines

5.1 The KIMath subroutines may be divided into four groups, Functions, Conversions, User Utilities and System Utilities. This section will deal with the first three classes.

5.1.1 The Function Subroutines are called ADD, SUB, MLTPLY, DIVIDE, SQRT, LOG, TENX, TANX, and ATANX. These routines may be divided into the Arithmetic Routines and the Transcendental Routines. Each of the routines requires that the argument registers (RX and RY) be in the Computational Format.

The following tables give some relevant information about the Arithmetic and Transcendental Routines:

Arithmetic Routines

Subroutine	Precision(1)	Error (2)	Symbolic Action
ADD	$0 < N \leq 16$	1 Count	RZ ← RX + RY (3)
SUB	$0 < N \leq 16$	1 Count	RZ ← RX - RY (3)
MLTPLY	$0 < N \leq 16$	1 Count	RZ ← RX * RY
DIVIDE	$0 < N \leq 16$	1 Count	RZ ← RX ÷ RY
SQRT	$0 < N \leq 16$	5 Counts	RZ ← $\sqrt{RX}$

(1) Note - Precision refers to the number of digits to which the calculation is to be carried out.

(2) Note - Error refers to the error in the least significant digit of the mantissa and so is the relative error.

(3) Note - In the case of ADD, at return time RX contains the argument with the largest absolute value. Thus one may use this routine to sort a table of numbers. In the case of SUB the sign of the mantissa of RY is changed and then the arguments are added. Thus the arguments are intact except for a sign.

Transcendental Routines

Subroutines	Interval	Precision	Error	Symbolic Action
LOG	$\left\{\frac{1}{\sqrt{10}}, \sqrt{10}\right\}$	14	$<10^{-8}$*	RX ← LOG_{10}(RX)
TENX	{0, 1}	12	10^{-8}	RX ← 10^{RX}
TANX	{0, 1}	14	10^{-8}	RX ← TAN(RX)
ATANX	{0, 1}	14	10^{-8}	RX ← ARCTAN(RX)

*Note: The error in the case of LOG is the absolute error; the others are relative error.

The user may use the bytes PREC and EXTRA together with the subroutine IPREC to calculate N, which is the value used by these routines. Note that the Transcendental Routines supply their own value for N and will overwrite any value supplied by the user.

Now that we know the names of the calculation routines and how to indicate to KIMath the desired precision of calculation we can add some statements to our sample program. Now it will look like (the > sign indicates new statements we have added):

```
                    ; (1) define storage
                    NDIG = 8
                       * = $5000
                    L  * = *+NDIG+3
                    C  * = *+NDIG+3
               TWOPI    .BYTE $00, $62, $83, $18, $54, $F0
               ONE      .BYTE $00, $10, $F0
                    ; (2) define precision
>                   EX = 2; use 2 extra digits in calculations
>                   LDA # NDIG
>                   STA   PREC
>                   LDA # EX
>                   STA   EXTRA
>                   JSR   IPREC ; calculate N
                    ; (3) read L and transfer to RX
```

```
                ; (4) read C and transfer to RY
                ; (5) compute L times C - answer stored in RZ
>                   JSR  MLTPLY ; do the multiplication
                ; (6) move RZ to RX and compute square root - adj. exp.
>                   JSR  SQRT
                ; (7) get 2π to RX
                ; (8) move RZ to RY and multiply
>                 JSR  MLTPLY
                ; (9) get 1 (constant) and put in RX
                ; (10) move RZ to RY and divide
>                 JSR  DIVIDE
                ; (11) move RZ to L and print out
                  END
```

Now by simply changing the assigned value for NDIG and EX and re-assembling the program, both calculated precision and storage allocation can be automatically changed. This allows re-assembly of the program with changed precision (and thus changed locations for registers L and C). The pointers will automatically be changed to the correct locations.

5.1.2 The Conversion Routines permit the user to transform data from the packed BCD, constant, or unpacked formats to the computational format and back again. They are PLOADX, PLOADY, ULOADX, ULOADY, PSTRES, and USTRES. These routines all use PREC to determine the number of bytes to be converted.

a) PLOADX, PLOADY: These routines load RX and RY with the packed format data found at the addresses {ARGYL, ARGXH} and {ARGYL, ARGYH}. The data is unpacked into the computational format and the value in PREC determines how many digits will be transferred. Note that PREC should be an even number to prevent the loss of a digit in transfer. This is natural when it is noted that the packed format requires an even number of digits in the mantissa bytes.

b) ULOADX, ULOADY: These routines load RX and RY with the unpacked format data found at the addresses {ARGXL, ARGYL} and {ARGYL, ARGYH} respectively. The data is converted into computational format and the number of digits transferred is determined by PREC. In this case PREC need not be even to effect an accurate transfer.

The use of ULOAD and PLOAD allows us to move data to the computational registers from memory registers. We can now add them to our program at the following locations:

```
                    ; (3) Read L and transfer to RX
> JSR GETNL         ; a user-provided subroutine to get a
>                   ; number from the terminal and place
>                   ; it in L in unpacked format.
> LDA #<L           ; low-order byte of address of L
> STA ARGXL
> LDA #>L           ; high-order byte
> STA ARGXH
> JSR ULOADX        ; move L to RX
                    ; (4) read C and transfer to RY
> LDA #<C           ; low-order byte of address of C
> STA ARGYL
> LDA  #>C          ; high-order byte of address of L
> STA ARGYH
> JSR GETNC         ; user-supplied routine to read value of C into
                      register C
> JSR ULOADY
```

Note: The assembler interprets the symbols "<L" to mean the low-order byte of the address of the variable named L. ">L" implies the high-order byte of the address of L.

c) PSTRES: This routine moves the contents of RB into the location specified by {RESL, RESH} as a starting address and converts it to packed format. The byte PREC is used to determine the number of bytes to be converted.

d) USTRES: This routine converts the contents of RB from computational format as it moves them to the destination specified by {RESL, RESH} and converts the contents to unpacked format. PREC, again, determines the number of bytes to be moved.

These routines allow transfer of the final answer to our example problem back to register L for printout:

```
            ; (11) move RØ to L and printout
>           LDA <L; set up
>           STA RESL; pointer
>           LDA >L; to locate
>           STA RESH; destination register
>           JSR USTRES; move it
>           JSR PRINTL; user-supplied routine to print the value
                in L
```

e) USRLKP: This routine loads RY with the constant format data beginning at the address {KONL, KONH}. The data is converted from constant format to computational format. The number of digits transferred is determined by the location of the byte containing an 'F.' This routine also uses the byte named NKON to point at the constant. Thus if the user sets NKON equal to zero and the pair {KONL, KONH} equal to some address prior to calling USRLKP then at return time RY contains the constant. {KONL, KONH, and NKON} point at the next constant, that is, {KONL, KONH} + NKON is the address of the byte following the byte containing an F (presumably the first byte of the next constant). Using this routine and POLY one may step through a table of coefficients and evaluate a polynomial. Use of POLY is covered in Appendix A. Clearly, one may also use this routine for other purposes such as a simple table lookup of constants needed frequently in a calculation.

Now we have the capability to load our constants into the program. Now we can fill out steps (7) and (9):

```
            ; (7) get 2π to RX
>           LDA <TWOPI; get low-order
>           STA KONL ; pointer byte
>           LDA >TWOPI; get high-order
>           STA KONH ; pointer byte
>           LDA #00
>           STA NKON ; initialize offset
>           JSR USRLKP ; lookup constant and put in RY
            ; still have to get RY to RX
```

```
                ; (9) get 1 (constant) and put in RX
>               JSR USRLKP ; a lot easier the second time
>               ; still must transfer RX from RY
```

Up to 256 bytes of constants may be indexed through by successive calls to USRLKP.

f) DECHEX: This routine converts the contents of the byte CNT from BCD to HEX. The converted value may be found at return time in CNT.

The User Utilities permit the user to move and clear various registers.

a) The routines whose names are of the form MV s, d, where "s" stands for source and "d" destination, allow the user to move the registers RX, RY, RØ, RM and RN about with ease. The following table shows the supported moves. RM and RN are working registers and will not normally be available to the user.

Source	Destinations	Corresponding Routine Names
RX	RY, RØ, RM, RN	MVXY, MVXØ, MVXM, MVXN
RY	RX, RØ, RM, RN	MVYX, MVYØ, MVYM, MVYN
RØ	RX, RY, RM, RN	MVØX, MVØY, MVØM, MVØN
RM	RX, RY, RØ, RN	MVMX, MVMY, MVMØ, MVMN
RN	RX, RY, RØ, RM	MVNX, MVNY, MVNØ, MVNM

b) The routines CLRX, CLRY and CLRØ may be used to set RX, RY and RØ equal to zero, respectively.

c) The routine XSY exchanges RX and RY.

The move subroutines allow us to transfer computational data between working registers. We need them in steps (6), (7), (8), (9), and (10):

```
                ; (6) Move RØ to RX and compute square root - adj. exp.
>               JSR MVØX; move RØ to RX
                JSR SQRT
                ; (7) get 2π to RX
                LDA #<TWOPI
```

```
                STA KONL
                LDA #>TWOPI
                STA KONH
                LDA #00
                STA NKON
                JSR USRLKP ; constant to RY
>               JSR MVYX ; move it to RX
                ; (8) move RØ to RY and multiply
>               JSR MVØY
                JSR MLTPLY
                ; (9) get 1 (constant) and put in RX
                JSR USRLKP ; get next constant
>               JSR MVYX ; move it to RX
                ; (10) move RØ to RY and divide
>               JSR MVØY ; move RØ to RY
                JSR DIVIDE
```

The sample program is now complete except for the fact that SQRT will only operate on numbers between 1 and 100, so we must adjust the exponent of the argument, take the square root and adjust the results. The two subroutines required are listed in Appendix B and are called SQRIN & SQROUT. They must be added to the program (they are not included in KIMath). Step (6) now becomes:

```
                ; (6) Move RØ to RX and compute square root
                JSR MVØX ; move RØ to RX
>               JSR SQRIN ; adjust exponent
                JSR SQRT ; compute root
>               JSR SQROUT ; adjust exponent back
```

5.2 Completed KIMath Program

```
                ; (1) define storage
                NDIG = 8
                   * = $5000
                L *  = *+NDIG+3
                C *  = *+NDIG+3
                TWOPI  .BYTE #00, $62, $83, $18, $54, $F0; TWOPI
                ONE    .BYTE $00, $10, $F0
                ; (2) define precision
```

```
EX = 2; use 2 extra digits in calculation
LDA #NDIG
STA PREC
LDA #EX
STA EXTRA
JSR PREC ; calculate N
; (3) Read L and transfer to RX
JSR GETNL; user-provided subroutine to input value for L
        ; to L register in unpacked format.
LDA #>L; low-order byte of address of L
STA ARGXL
LDA #<L ; high-order byte
STA ARGXH
JSR ULOADX ; Move L to RX
; (4) Read C and transfer to RY
LDA #<C ; low-order byte of address of C
STA ARGYL
LDA #>C ; high-order byte of address of C
STA ARGYH
JSR GETNC ; user-provided subroutine to input value
          ; for C to C register in unpacked format.
JSR ULOADY ; transfer C to RY
; (5) Compute L times C
JSR MLTPLY
; (6) Move RE to RX and compute square root
JSR MVEX ; move RE to RX
JSR SQRIN ; adjust exponent
JSR SQRT ; compute root
JSR SQROUT ; adjust exponent back
; (7) get 2pi to RX
LDA #<TWOPI ; address low of First constant
STA KONL
LDA #>TWOPI
STA KONH
LDA #00
```

```
STA NKON
JSR USRLKP ; constant to RY
JSR MVYX; move it to RX
; (8) Move RB to RY and multiply
JSR MVBY
JSR MLTPLY
; (9) get 1 (constant) and put in RX
JSR USRLKP ; get next constant
JSR MVYX ; move it to RX
; (10) Move RB to RY and divide
JSR MVBY ; move RB to RY
JSR DIVIDE ;
; (11) Move RB to L and print it out
LDA #<L ; set up
STA RES ; pointer
LDA #>L ; to locate
STA RES+1 ; destination register
JSR USTRES ; move it
JSR PRINTL ; user-supplied routine to print out L.
.END
```

Note: The above program would also require that the starting address of all KIMath subroutines be defined. See Appendix D for the correct addresses.

Appendix A

EVALUATING POLYNOMIALS

KIMath uses polynomial approximations to generate several of its functions, and these subroutines are also available for calculation of user-defined polynomials.

The user must first define the constants to be used in the calculation, then define the degree of the polynomial and the starting address of the stored constant. Finally, the argument for the polynomial is transferred to RX and the routine POLY is called. The value of the polynomial is returned in RB.

Specifically, given a polynomial of the form:

$$y = c_0 + c_1x + c_2x^2 + c_3x^3 + \cdots + c_Nx^N$$

the coefficients (c_0, c_1, c_2, c_3, ..., c_N) are stored in constant format in contiguous memory. The highest order coefficient is stored in the lowest memory address. NKON is set to zero, DEG is set to a value of N - 1 where N is the degree of the polynomial. The address pair (KONL, KONH) must point to the first byte of the constant storage area. RX is loaded with the value of x and the subroutine POLY is called. Upon return, y (the value of the polynomial) is in RB.

A Sample Program

The sine function can be expressed as:

$$\sum_{N=1}^{\infty} (-1)^{N+1} \frac{Z^{2N-1}}{(2N-1)!}$$

Expanded, this yields the polynomial:

$$\sin Z = Z - \frac{Z^3}{3!} + \frac{Z^5}{5!} - \frac{Z^7}{7!} + \frac{Z^9}{9!} - \frac{Z^{11}}{11!} \cdots$$

For our example we will use the first six non-zero terms for our application. Our coefficients are:

$$c_0 = 0$$

$$c_1 = 1.0$$

$$c_2 = 0$$

$$c_3 = -\frac{1}{3!} = -1.6666667\ E\text{-}1$$

$$c_4 = 0$$

$$c_5 = +\frac{1}{5!} = +8.3333333\ E\text{-}3$$

$$c_6 = 0$$

$$c_7 = -\frac{1}{7!} = -1.9841270\ E\text{-}4$$

$$c_8 = 0$$

$$c_9 = +\frac{1}{9!} = 2.7557319\ E\text{-}6$$

$$c_{10} = 0$$

$$c_{11} = -\frac{1}{11!} = -2.5052108\ E\text{-}8$$

A program to evaluate the sine function would look like:

```
                ; define parameter and pointers
          PREC * = $10
                   .BYTE 04 ; value for PREC
         EXTRA * = $11
                   .BYTE 04 ; value for EXTRA
           KON * = $0E
                   .WORD $3000
           DEG * = $05
                   .BYTE 10
          NCON * = $01
                   .BYTE $00
               * = $3000 ; starting address
             NIN = * + 7 ; reserve 7 bytes for input register
; define constants CØ - C11 - note that they are stored in
                              reverse order
                   .BYTE $C0,$25,$05,$21,$08,$F8 ; C11
                   .BYTE 0,0,$F0;C10
                   .BYTE $40,$27,$55,$73,$19,$F6 ; C9
                   .BYTE 0,0,$F0 ; C8
                   .BYTE $C0,$19,$84,$12,$70,$F4 ; C7
                   .BYTE 0,0,$F0 ; C6
                   .BYTE 04,$83,$33,$33,$33,$F3 ; C5
                   .BYTE 0,0,$F0 ; C4
```

```
              .BYTE $C0,$16,$66,$66,$67,$F1 ; C3
              .BYTE 0,0$F0 ; C2
              .BYTE 0,01,$F0 ; C1
       CONST .BYTE 0,0,$F0 ; CØ
; call user-written subroutine to load some
; value into NIN register.
              JSR GETVAL

LDA KON ; set up
STA ARGXL ; pointer to
LDA KON+1 ; input
STA ARGXH ; buffer

JSR ULOADX ; move NIN to RX and convert
JSR POLY ; evaluate polynomial
LDA KON ; set up
STA RES ; pointer to
LDA KON+1 ; output
STA RES+1 ; results
JSR  USTRES ; move RØ to NIN and convert

; call user-written subroutine to printout
; value in NIN
JSR PUTVAL
.END ; all done
```

Appendix B

APPLICATIONS

In this section we shall deal with extension of the range of the functions, and extension of the function set.

a) Common Logarithm: Let $x \cdot 10^r$ denote an arbitrary positive floating-point number. Then

$$x \cdot 10^r = x \cdot 10^{-1/2} \cdot 10^{1/2} \cdot 10^r$$

$$= (x \cdot 10^{-1/2}) \cdot 10^{r+1/2}$$

Now $1 \leq x < 10$ and so $10^{-1/2} \leq x \cdot 10^{-1/2} < 10^{1/2}$. But $\sqrt{0.1} = 10^{-1/2}$ and so LOG may be used to evaluate $\log_{10}(x \cdot 10^{-1/2})$. By using the following equation, one may compute the common log for any positive argument.

$$\log_{10}(x \cdot 10^r) = \log_{10}(x \cdot 10^{-1/2} \cdot 10^{r+1/2})$$

$$= \log_{10}(x \cdot 10^{-1/2}) + r + 0.5$$

b) Common Antilog: Let x be an arbitrary floating-point number satisfying the inequality

$$|x| < 100,$$

and let $I = [x]$ and $F = \langle x \rangle$, where $[x]$ stands for the largest integer less than or equal to x and $\langle x \rangle = x - [x]$. With these definitions in mind we find that

$$x = I + F$$

Now $0 \leq F < 1$ and I is an integer and so

$$10^x = 10^F \cdot 10^I$$

Since $0 \leq F < 1$ we may use TENX to find 10^F, after which we have a number between 1 and 10. Therefore I is the floating point exponent of 10^x.

c) Tangent: Let x be a floating-point number which satisfies the inequality $0 \leq x < \frac{\pi}{2}$. This implies that $0 \leq x \cdot \frac{2}{\pi} < 1$. Now $\tan\left(\frac{\pi}{4} \cdot \frac{2}{\pi} x\right) = \tan\left(\frac{x}{2}\right)$ but

$$\tan x = \frac{2 \tan\left(\frac{x}{2}\right)}{1 - \tan^2\left(\frac{x}{2}\right)}$$

Thus by applying TANX to the value of $\frac{2}{\pi} x$ and applying the above trigonometric identity one may extend the tangent function to the interval $[0, \frac{\pi}{2})$. Note also that

$$\cos x = \frac{1 - \tan^2\left(\frac{x}{2}\right)}{1 + \tan^2\left(\frac{x}{2}\right)} \qquad \text{and}$$

$$\sin x = \frac{2 \tan\left(\frac{x}{2}\right)}{1 + \tan^2\left(\frac{x}{2}\right)}$$

and so one may use TANX to evaluate the other trig functions.

d) Arc tangent: This function may be extended from the interval [0, 1] by means of the following identities:

$$\arctan(-x) = -\arctan x$$

$$\arctan(|x|) = \frac{\pi}{2} - \arctan\left(\frac{1}{|x|}\right), \qquad x \neq 0$$

With these identities and the ATANX subroutine one can evaluate the arc tangent function for any argument. The other inverse trigonometric functions may then be computed by means of the following identities:

$$\arcsin x = \arctan\left(\frac{x}{(1 - x^2)^{1/2}}\right)$$

$$\text{arc cos } x = \arctan \frac{(1 - x^2)^{1/2}}{x}$$

e) Square Root: Since every floating-point number can be expressed as $x \cdot 10^r$ where $1 \leq |x| < 100$ and $|r|$ is even or zero, one can form the square root as follows:

$$\sqrt{x \cdot 10^r} = 10^{r/2} \cdot \sqrt{x}$$

where $x \geq 0$ and $\sqrt{x}$ is evaluated by using SQRT. Note that $r/2$ is an integer and $1 \leq \sqrt{x} < 10$ and so $r/2$ is the floating-point exponent of $\sqrt{x \cdot 10^r}$.

Chapter 5 mentions that the sample program must be provided with two subroutines to extend the range of the square root function. These subroutines are provided below. The subroutine SQRIN must be called immediately prior to calling SQRT and SQROUT must be called immediately afterwards. SQRIN and SQROUT are not part of KIMath and must be supplied by the user:

Exponent Adjustment for SQRT

```
                VAL * = *+1
SQRIN           SED
                SEC
                LDA   EX
                SBC   #2
                STA   EX
                BCC   OUT
                CLC
                LDA   VAL
                ADC   #1
                STA   VAL
                BNE   SQ1
OUT             ADC   #2
                BIT   SX
                BVC   SQ2
                STA   EX
                CLC
                ADC   VAL
                STA   VAL
                RTS
```

```
SQ2     STA   EX
        RTS

SQROUT  LDA   VAL
        STA   EZ
        RTS
```

Appendix C

THE APPROXIMATIONS

This section is included for those users who are interested in how KIMath generates its functions. Understanding of this section is not required for use of KIMath.

1) LOG: This subroutine has been built around the rational function given by the following equation:

$$R(x) = t(x)\ p(t(x)),$$

where $t(x) = \frac{x - 1}{x + 1}$, $x \in [\sqrt{0.1}, \sqrt{10}]$ and $p(x)$ is a polynomial given by

$$p(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5$$

where

$$a_0 = 8.685887483405 \times 10^{-1}$$
$$a_1 = 2.89551130267 \times 10^{-1}$$
$$a_2 = 1.731095517 \times 10^{-1}$$
$$a_3 = 1.3136901121 \times 10^{-1}$$
$$a_4 = 5.53427387 \times 10^{-2}$$
$$a_5 = 1.820912997 \times 10^{-1}$$

The absolute error on the interval $[\sqrt{0.1}, \sqrt{10}]$ for $\log_{10}(x)$ as approximated by $R(x)$ is less than 1×10^{-8}.

2) TENX: This subroutine has been built around the polynomial given by the following equation:

$$P(x) = (a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7)^2,$$

where $x \in [0, 1]$ and

$$a_0 = 1$$
$$a_1 = 1.15129277603$$
$$a_2 = 6.6273088429 \times 10^{-1}$$
$$a_3 = 2.5439357484 \times 10^{-1}$$

$$a_4 = 7.295173666 \times 10^{-2}$$
$$a_5 = 1.742111988 \times 10^{-2}$$
$$a_6 = 2.55491796 \times 10^{-3}$$
$$a_7 = 9.3264267 \times 10^{-4}$$

The relative error on the interval [0, 1] for 10^x as approximated by P(x) is less than 5×10^{-9}.

3) TANX: This subroutine has been built around the polynomial given by

$$P(x) = x(a_0 + a_1(x^2)^1 + a_2(x^2)^2 + a_3(x^2)^3 + a_4(x^2)^4 + a_5(x^2)^5 + a_6(x^2)^6,$$

where

$$a_0 = 7.853981762291 \times 10^{-1}$$
$$a_1 = 1.614897776174 \times 10^{-1}$$
$$a_2 = 3.98659104705 \times 10^{-2}$$
$$a_3 = 9.8345945393 \times 10^{-3}$$
$$a_4 = 2.7974335037 \times 10^{-3}$$
$$a_5 = 2.031171084 \times 10^{-4}$$
$$a_6 = 4.109741948 \times 10^{-4}$$

The relative error on the interval [0, 1] for $\tan\left(\frac{\pi}{4} x\right)$ as approximated by P(x) is less than 1×10^{-8}.

4) ATANX: This subroutine has been built around the polynomial given by

$$P(x) = x(a_0 + a_1(x^2) + a_2(x^2)^2 + a_3(x^2)^3 + a_4(x^2)^4 + a_5(x^2)^5 + a_6(x^2)^6 + a_7(x^2)^7 + a_8(x^2)^8),$$

where

$$a_0 = 9.999999847657 \times 10^{-1}$$
$$a_1 = -3.333307334505 \times 10^{-1}$$
$$a_2 = 1.999261939166 \times 10^{-1}$$
$$a_3 = -1.420364446652 \times 10^{-1}$$
$$a_4 = 1.06409340253 \times 10^{-1}$$
$$a_5 = -7.50429453889 \times 10^{-2}$$
$$a_6 = 4.26915192711 \times 10^{-2}$$
$$a_7 = -1.60686289604 \times 10^{-2}$$
$$a_8 = 2.8498896208 \times 10^{-3}$$

The relative error on the interval [0, 1] for arctan x as approximated by P(x) is less than 1×10^{-8}.

APPENDIX D

KIMath Addresses

CALCULATION SUBROUTINES

ADDRESS	NAME
F808	ADD
F800	SUB
F90B	MULPLY
FA16	DIVIDE
FA9E	SQRT
FAE7	LOG
FB41	TENX
FB5C	TANX
FB78	ATANX

UTILITY ROUTINES

FEF0	SAVXY
FEF5	RCLXY
FEE8	IPREC
FE0A	PLOADX
FE23	PLOADY
FE8A	ULOADX
FEA2	ULOADY
FE3C	PSTRES
FEBA	USTRES
FD92	USRLKP
FDC1	POLY
FD71	CLRX
FD7C	CLRY
FD87	CLRZ
FBC3	DECHEX
FCBF	XSY

WORKING STORAGE

PAGE ZERO STARTING ADDRESS	NAME
10	PREC
11	EXTRA
06	ARGXL
07	ARGXH
08	ARGYL
09	ARGYH
0A	RESL
0B	RESH
0E	KONL
0F	KONH
01	NKON
05	DEG
03	CNT

COMPUTATION REGISTERS

STARTING ADDRESS	NAME
0235	RX
0235	SX (sign)
0246	EX (exponent)
0247	RY
0247	SY
0258	EY
0259	RZ
0259	SY
026A	EY

```
CARD =  LOC   CODE      CARD
     1  0000            *=$0000
     2
     3                  COPYRIGHT MUS TECHNOLOGY, INC. SEPT.,1976
     4                  ALL RIGHTS RESERVED.    REV. 0
     5
     6             LEN  =17
     7             XY   =$01
     8             XZ   =$02
     9             XM   =$03
    10             XN   =$04
    11             YX   =$10
    12             YZ   =$12
    13             YM   =$13
    14             YN   =$14
    15             ZX   =$20
    16             ZY   =$21
    17             ZM   =$23
    18             ZN   =$24
    19             MX   =$30
    20             MY   =$31
    21             MZ   =$32
    22             MN   =$34
    23             NX   =$40
    24             NY   =$41
    25             NZ   =$42
    26             NM   =$43
    27  0000       N    *=*+1
    28  0001       NKON *=*+1
    29  0002       J    *=*+1
    30  0003       CNT  *=*+1
    31  0004       LENGTH *=*
    32  0004       CNTA *=*+1
    33  0005       DEG  *=*+1
    34  0006       ARGXL *=*+1
    35  0007       ARGXH *=*+1
    36  0008       ARGYL *=*+1
    37  0009       ARGYH *=*+1
    38  000A       RES  *=*+2
    39  000C       PTR  *=*+2
    40  000E       KON  *=*+1
    41  000F       KONH *=*+1
    42  0010       PREC *=*+1
    43  0011       EXTRA *=*+1
    44  0012       TEMP *=*+1
    45  0013       TEMP1 *=*+1
    46  0014       OVERR *=*+1
    47  0015       TMPX *=*+1
    48  0016       TMPY *=*+1
    49  0017            *=$0200
    50  0200       RA   *=*+LEN+1
    51  0212       RB   *=*+LEN+1
    52  0224       RQ   *=*+LEN
```

```
CARD =  LOC      CODE          CARD
    53  0235                RX      *=*
    54  0235                SX      *=*+LEN
    55  0246                EX      *=*+1
    56  0247                RY      *=*
    57  0247                SY      *=*+LEN
    58  0258                EY      *=*+1
    59  0259                RZ      *=*+0
    60  0259                SZ      *=*+LEN
    61  026A                EZ      *=*+1
    62  026B                RM      *=*+LEN+1
    63  027D                RN      *=*+LEN+1
    64  028F                RAMCOD  *=*+3
    65  0292                RAMA    *=*+3
    66  0295                RAMB    *=*+5
    67  029A                        *=$F800
    68
    69                       FLOATING POINT ADD-SUBTRACT ROUTINE.
    70
    71  F800  AD 47 02    SUB     LDA SY
    72  F803  49 80               EOR =$80
    73  F805  8D 47 02            STA SY
    74  F808  AD 35 02    ADD     LDA SX
    75  F80B  4D 47 02            EOR SY
    76  F80E  85 12               STA TEMP
    77  F810  F8                  SED
    78
    79                       CLEAR WORKING STORAGE.
    80
    81  F811  20 B8 FB            JSR CLEAR
    82
    83                       TEST RX FOR ZERO.
    84
    85  F814  20 A6 FC            JSR XZTST
    86
    87                       TEST RY FOR ZERO.
    88
    89  F817  F0 0E               BEQ ADD2
    90  F819  20 B3 FC            JSR YZTST
    91  F81C  F0 0C               BEQ ADD3
    92  F81E  24 12               BIT TEMP
    93  F820  50 70               BVC ADD6
    94
    95                       IF THE SIGNS OF THE EXPONENTS
    96                       DIFFER THEN SWAP RX AND RY.
    97
    98  F822  2C 35 02    ADD1    BIT SX
    99  F825  50 03               BVC ADD3
   100  F827  20 BF FC    ADD2    JSR XSY
   101  F82A  F8          ADD3    SED
   102  F82B  24 12               BIT TEMP
   103  F82D  70 03               BVS ADD31
   104  F82F  4C AC F8            JMP ADD9
```

```
CARD =  LOC    CODE        CARD
   105  F832  AD 46 02  ADD31   LDA EX
   106  F835  18               CLC
   107  F836  6D 58 02          ADC EY
   108  F839  B0 12             BCS ADD5
   109  F83B  85 03      ADD4    STA CNT
   110
   111                     COMPUTE THE HEX VALUE OF THE
   112                     BCD DIFFERENCE OF THE EXPONENTS.
   113
   114  F83D  20 C3 FB          JSR DECHEX
   115  F840  C5 00             CMP N
   116  F842  B0 09             BCS ADD5
   117
   118                     MOVE RY TO RB.
   119
   120  F844  20 1C FC          JSR RBERY
   121
   122                     ALIGN DECIMAL POINTS.
   123
   124  F847  20 EE FB          JSR RSBCNT
   125
   126                     ROUND RB OFF.
   127
   128  F84A  20 07 FC          JSR RBOFF
   129  F84D  AD 46 02  ADD5    LDA EX
   130  F850  8D 6A 02          STA EZ
   131
   132                     MOVE RX TO RA.
   133
   134  F853  20 29 FC          JSR RAERX
   135  F856  24 12             BIT TEMP
   136  F858  30 6A             BMI ADD13
   137
   138                     ADD RB TO RA.
   139  F85A  20 5C FC          JSR RAPRB
   140  F85D  AD 00 02          LDA RA
   141  F860  F0 16             BEQ ADD120
   142  F862  20 A6 FB          JSR RSRA
   143
   144                     CORRECT SIGN AND EXPONENT.
   145
   146  F865  AD 46 02          LDA EX
   147  F868  38                SEC
   148  F869  2C 35 02          BIT SX
   149  F86C  50 19             BVC ADD110
   150  F86E  E9 01             SBC =1
   151  F870  8D 6A 02          STA EZ
   152  F873  D0 08             BNE ADD120
   153  F875  A9 BF             LDA =$BF
   154  F877  2D 35 02          AND SX
   155  F87A  4C 80 F8          JMP ADD12
   156  F87D  AD 35 02  ADD120  LDA SX
```

```
CARD =  LOC    CODE        CARD
   157  F880  8D 59 02  ADD12   STA SZ
   158
   159                    MOVE RA TO RZ.
   160
   161  F883  20 4F FC  ADD121  JSR RZERA
   162  F886  60                RTS
   163  F887  69 00     ADD110  ADC =0
   164  F889  8D 6A 02          STA EZ
   165  F88C  90 EF              BCC ADD120
   166
   167                    SET RZ=9.9...9E99
   168
   169  F88E  20 D2 FC          JSR INFIN
   170  F891  60                RTS
   171
   172                    COMPARE ABS(RX) TO ABS(RY).
   173
   174  F892  20 82 FC  ADD6    JSR COMPXY
   175  F895  A5 04              LDA CNTA
   176  F897  F0 03              BEQ ADD8
   177
   178                    SWAP RX AND RY,
   179                    SO THAT RX HAS THE
   180                    LARGEST ABS. VALUE.
   181
   182  F899  20 BF FC  ADD7    JSR XSY
   183  F89C  AD 46 02  ADD8    LDA EX
   184  F89F  CD 58 02          CMP EY
   185  F8A2  F0 05              BEQ ADD81
   186  F8A4  90 F3              BCC ADD7
   187  F8A6  4C 22 F8          JMP ADD1
   188  F8A9  4C 2A F8  ADD81   JMP ADD3
   189
   190                    COMPUTE THE ABSOLUTE VALUE
   191                    OF THE SIGNED DIFFERENCE OF
   192                    THE EXPONENTS.
   193
   194  F8AC  38        ADD9    SEC
   195  F8AD  2C 35 02          BIT SX
   196  F8B0  70 09              BVS ADD10
   197  F8B2  AD 46 02          LDA EX
   198  F8B5  ED 58 02          SBC EY
   199  F8B8  4C 3B F8          JMP ADD4
   200  F8BB  AD 58 02  ADD10   LDA EY
   201  F8BE  ED 46 02          SBC EX
   202  F8C1  4C 3B F8          JMP ADD4
   203
   204                    SUBTRACT RB FROM RA.
   205
   206  F8C4  20 70 FC  ADD13   JSR RAMRB
   207  F8C7  AD 46 02          LDA EX
   208  F8CA  8D 6A 02          STA EZ
```

```
CARD =  LOC    CODE        CARD
  209   F8CD  AD 35 02          LDA SX
  210   F8D0  8D 59 02          STA SZ
  211
  212                           TEST RA FOR ZERO.
  213
  214   F8D3  20 99 FC          JSR AZTST
  215   F8D6  F0 18             BEQ ADD18
  216   F8D8  AD 01 02  ADD15   LDA RA+1
  217   F8DB  D0 A6             BNE ADD121
  218
  219                           IF RA+1 IS ZERO THEN
  220                           LEFT SHIFT RA ONE DIGIT.
  221
  222   F8DD  20 91 FB          JSR LSRA
  223   F8E0  2C 59 02  ADD17   BIT SZ
  224   F8E3  38                SEC
  225   F8E4  AD 6A 02          LDA EZ
  226   F8E7  50 08             BVC ADD20
  227   F8E9  69 00             ADC =0
  228   F8EB  8D 6A 02          STA EZ
  229   F8EE  90 E8             BCC ADD15
  230
  231                           SET RZ EQUAL TO ZERO.
  232
  233   F8F0  20 87 FD  ADD18   JSR CLRZ
  234   F8F3  60        ADD19   RTS
  235
  236                           ADJUST SIGN AND EXPONENT
  237                           OF THE ANSWER.
  238
  239   F8F4  E9 01     ADD20   SBC =1
  240   F8F6  8D 6A 02          STA EZ
  241   F8F9  B0 DD             BCS ADD15
  242   F8FB  A9 01             LDA =1
  243   F8FD  8D 6A 02          STA EZ
  244   F900  A9 40             LDA =$40
  245   F902  0D 59 02          ORA SZ
  246   F905  8D 59 02          STA SZ
  247   F908  4C D8 F8          JMP ADD15
  248
  249                           FLOATING POINT PRODUCT ROUTINE.
  250
  251   F908  F8        MLTPLY  SED
  252
  253                           CLEAR WORKING STORAGE.
  254
  255   F90C  20 B8 FB          JSR CLEAR
  256   F90F  A9 00             LDA =0
  257   F911  85 03             STA CNT
  258   F913  85 13             STA TEMP1
  259
  260                           TEST RA FOR ZERO.
```

```
CARD =  LOC    CODE        CARD
   261
   262  F915  20 A6 FC          JSR XZTST
   263  F918  F0 05             BEQ MULT1
   264
   265                     TEST RY FOR ZERO.
   266
   267  F91A  20 B3 FC          JSR YZTST
   268  F91D  D0 08             BNE MULT3
   269
   270                     SET RZ EQUAL TO ZERO.
   271
   272  F91F  20 87 FD  MULT1   JSR CLRZ
   273  F922  60                RTS
   274
   275                     MOVE RA TO RZ.
   276
   277  F923  20 4F FC  MULT2   JSR RZERA
   278  F926  60                RTS
   279
   280                     MOVE RY TO RB.
   281
   282  F927  20 1C FC  MULT3   JSR RBERY
   283
   284                     MOVE RX TO RQ.
   285
   286  F92A  20 36 FC          JSR RQERX
   287
   288                     FORM PRODUCT OF MANTISSAS.
   289
   290  F92D  20 53 FA          JSR MLT
   291
   292                     FIGURE THE SIGN AND EXPONENT OF
   293                     OF THE ANSWER FOR THE MULTIPLY
   294                     AND DIVIDE ROUTINES.
   295
   296  F930  AD 47 02  MULT4   LDA SY
   297  F933  4D 35 02          EOR SX
   298  F936  85 12             STA TEMP
   299  F938  24 12             BIT TEMP
   300  F93A  AD 46 02          LDA EX
   301  F93D  70 1E             BVS MD100
   302  F93F  18        MD1     CLC
   303  F940  6D 58 02          ADC EY
   304  F943  90 2B             BCC MD2
   305  F945  D0 61             BNE MD59
   306  F947  A5 13             LDA TEMP1
   307  F949  F0 15             BEQ MDOV2
   308  F94B  2C 35 02          BIT SX
   309  F94E  70 67             BVS MD7
   310  F950  A5 04             LDA CNTA
   311  F952  F0 5F             BEQ MD61
   312  F954  A9 00             LDA =0
```

```
CARD =  LOC     CODE          CARD
   313  F956  85 04          STA CNTA
   314  F958  A9 99    MDOV1 LDA =$99
   315  F95A  4C 70 F9       JMP MD2
   316  F95D  4C C3 F9 MD100 JMP MD10
   317  F960  2C 35 02 MDOV2 BIT SX
   318  F963  50 4E          BVC MD61
   319  F965  AD 00 02       LDA RA
   320  F968  F0 EE          BEQ MDOV1
   321  F96A  20 A6 F8       JSR RSRA
   322  F96D  4C 58 F9       JMP MDOV1
   323  F970  8D 6A 02 MD2   STA EZ
   324  F973  D0 64          BNE MD11
   325  F975  AD 35 02       LDA SX
   326  F978  29 BF          AND =$BF
   327  F97A  8D 59 02 MD3   STA SZ
   328  F97D  A5 12    MD4   LDA TEMP
   329  F97F  30 3A          BMI MD8
   330  F981  A9 7F          LDA =$7F
   331  F983  2D 59 02       AND SZ
   332  F986  8D 59 02 MD5   STA SZ
   333  F989  A5 13          LDA TEMP1
   334  F98B  D0 52          BNE DIVEXT
   335  F98D  AD 00 02       LDA RA
   336  F990  F0 13          BEQ MD51
   337  F992  20 A6 F8       JSR RSRA
   338  F995  AD 6A 02       LDA EZ
   339  F998  2C 59 02       BIT SZ
   340  F99B  70 6B          BVS MD9
   341  F99D  18             CLC
   342  F99E  69 01          ADC =1
   343  F9A0  F0 0C          BEQ MD6
   344  F9A2  8D 6A 02       STA EZ
   345  F9A5  4C 23 F9 MD51  JMP MULT2
   346  F9A8  AD 35 02 MD59  LDA SX
   347  F9AB  8D 59 02       STA SZ
   348  F9AE  2C 59 02 MD6   BIT SZ
   349  F9B1  70 04          BVS MD7
   350  F9B3  20 D2 FC MD61  JSR INFIN
   351  F9B6  60             RTS
   352  F9B7  20 87 FD MD7   JSR CLRZ
   353  F9BA  60             RTS
   354  F9BB  A9 80    MD8   LDA =$80
   355  F9BD  0D 59 02       ORA SZ
   356  F9C0  4C 86 F9       JMP MD5
   357  F9C3  38       MD10  SEC
   358  F9C4  ED 58 02       SBC EY
   359  F9C7  B0 A7          BCS MD2
   360  F9C9  38             SEC
   361  F9CA  AD 58 02       LDA EY
   362  F9CD  ED 46 02       SBC EX
   363  F9D0  8D 6A 02       STA EZ
   364  F9D3  AD 47 02       LDA SY
```

```
CARD =  LOC    CODE        CARD
   365  F9D6  4C 7A F9           JMP MD3
   366  F9D9  AD 35 02  MD11     LDA SX
   367  F9DC  4C 7A F9           JMP MD3
   368  F9DF  A5 04     DIVEXT   LDA CNTA
   369  F9E1  F0 C2              BEQ MD51
   370  F9E3  2C 59 02  DVEXT0   BIT SZ
   371  F9E6  AD 6A 02           LDA E7
   372  F9E9  38                 SEC
   373  F9EA  50 0A              BVC DVEXT2
   374  F9EC  69 00              ADC =0
   375  F9EE  F0 BE              BEQ MD6
   376  F9F0  8D 6A 02  DVEXT1   STA E7
   377  F9F3  4C 23 F9           JMP MULT2
   378  F9F6  F0 05     DVEXT2   BEQ DVEXT3
   379  F9F8  E9 01              SBC =1
   380  F9FA  4C F0 F9           JMP DVEXT1
   381  F9FD  AD 59 02  DVEXT3   LDA SZ
   382  FA00  09 40              ORA =$40
   383  FA02  8D 59 02           STA SZ
   384  FA05  4C E3 F9           JMP DVEXT0
   385  FA08  38        MD9      SEC
   386  FA09  E9 01              SBC =1
   387  FA0B  F0 06              BEQ MD22
   388  FA0D  8D 6A 02           STA E7
   389  FA10  4C 23 F9           JMP MULT2
   390  FA13  4C 70 F9  MD22     JMP MD2
   391
   392                       FLOATING POINT DIVIDE ROUTINE.
   393
   394  FA16  F8         DIVIDE SED
   395
   396                       TEST RY FOR ZERO.
   397
   398  FA17  20 B3 FC           JSR YZTST
   399  FA1A  F0 97              BEQ MD61
   400
   401                       TEST RX FOR ZERO.
   402
   403  FA1C  20 A6 FC           JSR XZTST
   404  FA1F  F0 96              BEQ MD7
   405
   406                       CLEAR WORKING STORAGE.
   407
   408  FA21  20 B8 FB           JSR CLEAR
   409
   410                       MOVE RX TO RA.
   411
   412  FA24  20 29 FC           JSR RAERX
   413
   414                       MOVE RY TO RB.
   415
   416  FA27  20 1C FC           JSR RBERY
```

```
CARD =  LOC      CODE        CARD
  417
  418                        COMPARE RX TO RY.
  419
  420   FA2A   20 82 FC            JSR COMPXY
  421
  422                        FORM QUOTIENT.
  423
  424   FA2D   20 75 FA            JSR DIV
  425
  426                        COMPUTE SIGN AND EXPONENT OF
  427                        ANSWER.
  428
  429   FA30   A9 01         DIV6  LDA =1
  430   FA32   85 13               STA TEMP1
  431   FA34   AD 47 02            LDA SY
  432   FA37   49 40               EOR =$40
  433   FA39   8D 47 02            STA SY
  434   FA3C   20 43 FC            JSR RAERQ
  435   FA3F   AD 01 02            LDA RA+1
  436   FA42   D0 03               BNE DIV7
  437   FA44   20 91 FB            JSR LSRA
  438   FA47   20 30 F9     DIV7   JSR MULT4
  439   FA4A   AD 47 02            LDA SY
  440   FA4D   49 40               EOR =$40
  441   FA4F   8D 47 02            STA SY
  442   FA52   60                  RTS
  443
  444                        THIS ROUTINE COMPUTES THE
  445                        PRODUCT OF THE MANTISSAS
  446                        OF THE ARGUMENTS BY REPEATED
  447                        ADDITION. THE RESULT IS BUILT
  448                        IN RA.
  449
  450   FA53   A5 00         MLT   LDA N
  451   FA55   85 02               STA J
  452   FA57   C6 02               DEC J
  453   FA59   A6 02         MLT0  LDX J
  454   FA5B   BD 24 02            LDA RQ,X
  455   FA5E   85 03               STA CNT
  456   FA60   C6 03         MLT1  DEC CNT
  457   FA62   30 06               BMI MLT2
  458   FA64   20 5C FC            JSR RAPRB
  459   FA67   4C 60 FA            JMP MLT1
  460   FA6A   20 A6 FB      MLT2  JSR RSRA
  461   FA6D   C6 02               DEC J
  462   FA6F   10 E8               BPL MLT0
  463   FA71   20 91 FB            JSR LSRA
  464   FA74   60                  RTS
  465
  466                        THIS ROUTINE COMPUTES THE
  467                        QUOTIENT OF RA AND RB BY
  468                        REPEATED SUBTRACTION. THE
```

```
CARD = LOC      CODE        CARD
   469                      RESULT IS BUILT IN RQ.
   470
   471  FA75  A9 00      DIV     LDA =0
   472  FA77  85 02              STA J
   473  FA79  A9 00      DIV0    LDA =0
   474  FA7B  85 03              STA CNT
   475  FA7D  20 7D FC   DIV1    JSR RAMRB
   476  FA80  90 04              BCC DIV2
   477  FA82  E6 03              INC CNT
   478  FA84  D0 F7              BNE DIV1
   479  FA86  20 5C FC   DIV2    JSR RAPRB
   480  FA89  20 91 FB           JSR LSRA
   481  FA8C  A6 02              LDX J
   482  FA8E  A5 03              LDA CNT
   483  FA90  9D 24 02           STA RQ,X
   484  FA93  E6 02              INC J
   485  FA95  A5 02              LDA J
   486  FA97  C5 00              CMP N
   487  FA99  F0 DE              BEQ DIV0
   488  FA9B  90 DC              BCC DIV0
   489  FA9D  60                 RTS
   490
   491                       THIS ROUTINE COMPUTES THE
   492                       SQUARE ROOT OF A FLOATING POINT
   493                       NUMBER BETWEEN 1 AND 100 BY
   494                       HERONS METHOD.
   495
   496  FA9E  A9 07      SQRT    LDA =7
   497  FAA0  85 01              STA AKON
   498  FAA2  20 F8 FC           JSR MVXN
   499  FAA5  20 87 FD           JSR CLRZ
   500  FAA8  A9 07              LDA =7
   501  FAAA  8D 5A 02           STA RZ+1
   502  FAAD  A9 08              LDA =8
   503  FAAF  8D 58 02           STA RZ+2
   504  FAB2  20 14 FD           JSR MVZM
   505  FAB5  20 20 FD   SQRT0   JSR MVMY
   506  FAB8  20 2C FD           JSR MVNX
   507  FABB  20 16 FA           JSR DIVIDE
   508  FABE  20 10 FD           JSR MVZY
   509  FAC1  20 1C FD           JSR MVMX
   510  FAC4  20 08 F8           JSR ADD
   511  FAC7  20 0C FD           JSR MVZX
   512  FACA  20 7C FD           JSR CLRY
   513  FACD  A9 40              LDA =$40
   514  FACF  8D 47 02           STA RY
   515  FAD2  A9 05              LDA =5
   516  FAD4  8D 48 02           STA RY+1
   517  FAD7  A9 01              LDA =1
   518  FAD9  8D 58 02           STA EY
   519  FADC  20 0B F9           JSR MLTPLY
   520  FADF  20 14 FD           JSR MVZM
```

```
CARD =  LOC     CODE        CARD
  521   FAE2  C6 01             DEC NKON
  522   FAE4  10 CF             BPL SQRTO
  523   FAE6  60                RTS
  524
  525                    THIS ROUTINE COMPUTES THE
  526                    COMMON LOG OF A FLOATING POINT
  527                    NUMBER BETWEEN SQRT(.1)AND SQRT(10).
  528
  529   FAE7  A9 0E      LOG   LDA =14
  530   FAE9  85 00            STA N
  531   FAEB  20 FC FE         JSR SETKON
  532   FAEE  20 F8 FC         JSR MVXN
  533   FAF1  20 7C FD         JSR CLRY
  534   FAF4  A9 01            LDA =1
  535   FAF6  8D 48 02         STA RY+1
  536   FAF9  20 00 F8         JSR SUB
  537   FAFC  20 2C FD         JSR MVNX
  538   FAFF  20 7C FD         JSR CLRY
  539   FB02  A9 01            LDA =1
  540   FB04  8D 48 02         STA RY+1
  541   FB07  20 18 FD         JSR MVZN
  542   FB0A  20 08 F8         JSR ADD
  543   FB0D  20 10 FD         JSR MVZY
  544   FB10  20 2C FD         JSR MVNX
  545   FB13  20 16 FA         JSR DIVIDE
  546   FB16  20 18 FD         JSR MVZN
  547   FB19  20 0C FD         JSR MVZX
  548   FB1C  20 10 FD         JSR MVZY
  549   FB1F  20 0B F9         JSR MLTPLY
  550   FB22  A9 04            LDA =4
  551   FB24  85 05            STA DEG
  552   FB26  A9 00            LDA =0
  553   FB28  85 01      LOGEND STA NKON
  554   FB2A  20 C1 FD         JSR POLY
  555   FB2D  20 30 FD         JSR MVNY
  556   FB30  20 0C FD   LGNDO  JSR MVZX
  557   FB33  20 0B F9         JSR MLTPLY
  558   FB36  A9 00      CHCP  LDA =0
  559   FB38  A2 07            LDX =LEN/2-1
  560   FB3A  9D 62 02   CHOPO STA RZ+9,X
  561   FB3D  CA               DEX
  562   FB3E  10 FA            BPL CHOPO
  563   FB40  60               RTS
  564
  565                    THIS ROUTINE COMPUTES THE
  566                    COMMON ANTI-LOG OF A FLOATING
  567                    POINT NUMBER BETWEEN 0 AND 1.
  568
  569   FB41  A9 0C      TENX  LDA #12
  570   FB43  85 00            STA N
  571   FB45  20 FC FE         JSR SETKON
  572   FB48  20 F0 FC         JSR MVXZ
```

```
CARD =  LOC   CODE        CARD
   573  FB4B  A9 06           LDA =6
   574  FB4D  85 05           STA DEG
   575  FB4F  A9 2E           LDA =46
   576  FB51  85 01           STA NKON
   577  FB53  20 C1 FD        JSR POLY
   578  FB56  20 10 FD        JSR MVZY
   579  FB59  4C 30 FB        JMP LGND0
   580
   581                    THIS ROUTINE COMPUTES THE
   582                    TANGENT OF A FLOATING POINT NUMBER
   583                    BETWEEN 0 AND PI/4.
   584
   585  FB5C  A9 0E      TANX  LDA =14
   586  FB5E  85 00           STA N
   587  FB60  20 FC FE        JSR SETKON
   588  FB63  20 F8 FC        JSR MVXN
   589  FB66  20 EC FC        JSR MVXY
   590  FB69  20 08 F9        JSR MLTPLY
   591  FB6C  20 36 FB        JSR CHOP
   592  FB6F  A9 05           LDA =5
   593  FB71  85 05           STA DEG
   594  FB73  A9 64           LDA =100
   595  FB75  4C 28 FB        JMP LOGEND
   596
   597                    THIS ROUTINE COMPUTES THE
   598                    ARCTANGENT OF A FLOATING POINT NUMBER
   599                    BETWEEN 0 AND 1.
   600
   601  FB78  A9 0E      ATANX LDA =14
   602  FB7A  85 00           STA N
   603  FB7C  20 FC FE        JSR SETKON
   604  FB7F  20 F8 FC        JSR MVXN
   605  FB82  20 EC FC        JSR MVXY
   606  FB85  20 08 F9        JSR MLTPLY
   607  FB88  A9 07           LDA =7
   608  FB8A  85 05           STA DEG
   609  FB8C  A9 9C           LDA =156
   610  FB8E  4C 28 FB        JMP LOGEND
   611
   612                    LEFT SHIFT RA ONE DIGIT.
   613
   614  FB91  A2 00      LSRA  LDX =0
   615  FB93  BD 01 02   LSRA0 LDA RA+1,X
   616  FB96  9D 00 02         STA RA,X
   617  FB99  E8               INX
   618  FB9A  E4 00            CPX N
   619  FB9C  90 F5            BCC LSRA0
   620  FB9E  F0 F3            BEQ LSRA0
   621  FBA0  A9 00            LDA =0
   622  FBA2  9D 00 02         STA RA,X
   623  FBA5  60               RTS
   624
```

```
  625                          RIGHT SHIFT RA ONE DIGIT.
  626
  627  FBA6  A6 00       RSRA   LDX N
  628  FBA8  CA                 DEX
  629  FBA9  BD 00 02    RSRA0  LDA RA,X
  630  FBAC  9D 01 02           STA RA+1,X
  631  FBAF  CA                 DEX
  632  FBB0  10 F7              BPL RSRA0
  633  FBB2  A9 00              LDA =0
  634  FBB4  8D 00 02           STA RA
  635  FBB7  60                 RTS
  636
  637                          CLEAR WORKING STORAGE.
  638
  639  FBB8  A2 34       CLEAR  LDX =LEN*3+1
  640  FBBA  A9 00              LDA =0
  641  FBBC  9D 00 02    AZ0    STA RA,X
  642  FBBF  CA                 DEX
  643  FBC0  10 FA              BPL AZ0
  644  FBC2  60                 RTS
  645
  646                          CONVERT THE CONTENTS OF CNT
  647                          FROM BCD TO HEX AND STORE THE
  648                          RESULT IN CNT.
  649
  650  FBC3  F8          DECHEX SED
  651  FBC4  A2 00              LDX =0
  652  FBC6  38                 SEC
  653  FBC7  A5 03       DHCNV1 LDA CNT
  654  FBC9  E9 16              SBC =$16
  655  FBCB  90 06              BCC DHCNV2
  656  FBCD  85 03              STA CNT
  657  FBCF  E8                 INX
  658  FBD0  4C C7 FB           JMP DHCNV1
  659  FBD3  D8          DHCNV2 CLD
  660  FBD4  A5 03              LDA CNT
  661  FBD6  C9 0A              CMP =$0A
  662  FBD8  90 04              BCC DHCNV3
  663  FBDA  29 0F              AND =$0F
  664  FBDC  69 09              ADC =$09
  665  FBDE  86 03       DHCNV3 STX CNT
  666  FBE0  06 03              ASL CNT
  667  FBE2  06 03              ASL CNT
  668  FBE4  06 03              ASL CNT
  669  FBE6  06 03              ASL CNT
  670  FBE8  05 03              ORA CNT
  671  FBEA  85 03              STA CNT
  672  FBEC  F8                 SED
  673  FBED  60          DHCNVE RTS
  674
  675                          RIGHT SHIFT RB CNT TIMES.
  676
```

```
CARD =  LOC     CODE        CARD
   677  FBEE  A5 03      RSBCNT LDA CNT
   678  FBF0  F0 29             BEQ RBOFF
   679  FBF2  A6 00             LDX N
   680  FBF4  BD 12 02   RSBC   LDA RB,X
   681  FBF7  9D 13 02          STA RB+1,X
   682  FBFA  CA                DEX
   683  FBFB  10 F7             BPL RSBC
   684  FBFD  A9 00             LDA =0
   685  FBFF  8D 12 02          STA RB
   686  FC02  C6 03             DEC CNT
   687  FC04  D0 E8             BNE RSBCNT
   688  FC06  60                RTS
   689
   690                       ROUND RB OFF.
   691
   692  FC07  A6 00      RBOFF  LDX N
   693  FC09  BD 13 02          LDA RB+1,X
   694  FC0C  C9 05             CMP =5
   695  FC0E  BD 12 02   RBOF   LDA RB,X
   696  FC11  69 90             ADC =$90
   697  FC13  29 0F             AND =$0F
   698  FC15  9D 12 02          STA RB,X
   699  FC18  CA                DEX
   700  FC19  10 F3             BPL RBOF
   701  FC1B  60         RBOFE  RTS
   702
   703                       MOVE RY TO RB.
   704
   705  FC1C  A6 00      RBERY  LDX N
   706  FC1E  CA                DEX
   707  FC1F  BD 48 02   RBRY   LDA RY+1,X
   708  FC22  9D 13 02          STA RB+1,X
   709  FC25  CA                DEX
   710  FC26  10 F7             BPL RBRY
   711  FC28  60                RTS
   712
   713                       MOVE RX TO RA.
   714
   715  FC29  A6 00      RAERX  LDX N
   716  FC2B  CA                DEX
   717  FC2C  BD 36 02   RARX0  LDA RX+1,X
   718  FC2F  9D 01 02          STA RA+1,X
   719  FC32  CA                DEX
   720  FC33  10 F7             BPL RARX0
   721  FC35  60         RARXE  RTS
   722
   723                       MOVE RX TO RQ.
   724
   725  FC36  A6 00      RQERX  LDX N
   726  FC38  CA                DEX
   727  FC39  BD 36 02   RQRX   LDA RX+1,X
   728  FC3C  9D 24 02          STA RQ,X
```

```
CARD =  LOC     CODE        CARD
   729  FC3F  CA              DEX
   730  FC40  10 F7           BPL RQRX
   731  FC42  60              RTS
   732
   733                      MOVE RQ TO RA.
   734
   735  FC43  A6 00     RAERQ LDX N
   736  FC45  BD 24 02  RARQ  LDA RQ,X
   737  FC48  9D 01 02        STA RA+1,X
   738  FC4B  CA              DEX
   739  FC4C  10 F7           BPL RARQ
   740  FC4E  60              RTS
   741
   742                      MOVE RA TO RZ.
   743
   744  FC4F  A6 00     RZERA LDX N
   745  FC51  CA              DEX
   746  FC52  BD 01 02  RZRAQ LDA RA+1,X
   747  FC55  9D 5A 02        STA RZ+1,X
   748  FC58  CA              DEX
   749  FC59  10 F7           BPL RZRAQ
   750  FC5B  60        RZRAE RTS
   751
   752                      ADD RB TO RA.
   753
   754  FC5C  A6 00     RAPRB LDX N
   755  FC5E  18              CLC
   756  FC5F  BD 00 02  APB   LDA RA,X
   757  FC62  7D 12 02        ADC RB,X
   758  FC65  69 90           ADC =$90
   759  FC67  29 0F           AND =$0F
   760  FC69  9D 00 02        STA RA,X
   761  FC6C  CA              DEX
   762  FC6D  10 F0           BPL APB
   763  FC6F  60              RTS
   764
   765                      SUBTRACT RB FROM RA.
   766
   767  FC70  A6 00     RAMRB LDX N
   768  FC72  38              SEC
   769  FC73  BD 00 02  AMB   LDA RA,X
   770  FC76  FD 12 02        SBC RB,X
   771  FC79  29 0F           AND =$0F
   772  FC7B  9D 00 02        STA RA,X
   773  FC7E  CA              DEX
   774  FC7F  10 F2           BPL AMB
   775  FC81  60              RTS
   776
   777                      COMPARE RX TO RY.
   778
   779  FC82  A9 00     COMPXY LDA =0
   780  FC84  85 04            STA CNTA
```

```
CARD =  LOC      CODE          CARD
   781  FC86  A6 00           LDX N
   782  FC88  CA              DEX
   783  FC89  38              SEC
   784  FC8A  BD 36 02  COM1   LDA RX+1,X
   785  FC8D  FD 48 02         SBC RY+1,X
   786  FC90  CA              DEX
   787  FC91  10 F7           BPL COM1
   788  FC93  90 01           BCC COM2
   789  FC95  60              RTS
   790  FC96  E6 04    COM2   INC CNTA
   791  FC98  60              RTS
   792
   793                        TEST RA FOR ZERO.
   794
   795  FC99  A6 00    AZTST  LDX N
   796  FC9B  E8              INX
   797  FC9C  BD 00 02 AZTST0 LDA RA,X
   798  FC9F  D0 11           BNE XZTST1
   799  FCA1  CA              DEX
   800  FCA2  10 F8           BPL AZTST0
   801  FCA4  30 0A           BMI XZTST2
   802
   803                        TEST RX FOR ZERO.
   804
   805  FCA6  A6 00    XZTST  LDX N
   806  FCA8  BD 35 02 XZTST0 LDA RX,X
   807  FCAB  D0 05           BNE XZTST1
   808  FCAD  CA              DEX
   809  FCAE  10 F8           BPL XZTST0
   810  FCB0  A9 00    XZTST2 LDA =0
   811  FCB2  60       XZTST1 RTS
   812
   813                        TEST RY FOR ZERO.
   814
   815  FCB3  A6 00    YZTST  LDX N
   816  FCB5  BD 47 02 YZTST0 LDA RY,X
   817  FCB8  D0 F8           BNE XZTST1
   818  FCBA  CA              DEX
   819  FCBB  10 F8           BPL YZTST0
   820  FCBD  30 F1           BMI XZTST2
   821
   822                        SWAP RX AND RY.
   823
   824  FCBF  A2 11    XSY    LDX =LEN
   825  FCC1  BD 35 02 XSY1   LDA RX,X
   826  FCC4  BC 47 02        LDY RY,X
   827  FCC7  9D 47 02        STA RY,X
   828  FCCA  98              TYA
   829  FCCB  9D 35 02        STA RX,X
   830  FCCE  CA              DEX
   831  FCCF  10 F0           BPL XSY1
   832  FCD1  60              RTS
```

```
CARD =  LOC      CODE        CARD
   833
   834                       SET RZ=9.9...9E99 AND OVERR=1.
   835
   836  FCD2  A6 00      INFIN   LDX N
   837  FCD4  CA                  DEX
   838  FCD5  A9 09               LDA =9
   839  FCD7  9D 5A 02    INFO   STA RZ+1,X
   840  FCDA  CA                  DEX
   841  FCDB  10 FA               BPL INFO
   842  FCDD  A9 99               LDA =$99
   843  FCDF  8D 6A 02            STA EZ
   844  FCE2  A9 00               LDA =0
   845  FCE4  8D 59 02            STA SZ
   846  FCE7  A9 01               LDA =1
   847  FCE9  85 14               STA OVERR
   848  FCEB  60                  RTS
   849
   850                       THE FOLLOWING ROUTINES ARE USED
   851                       TO MOVE THE CONTENTS FROM ONE
   852                       REGISTER TO ANOTHER. THE NAMES ARE
   853                       OF THE FORM MVSD, WHERE S STANDS
   854                       FOR SOURCE AND D FOR DESTINATION.
   855
   856  FCEC  A9 01       MVXY    LDA =XY
   857  FCEE  D0 4A               BNE MVTR
   858  FCF0  A9 02       MVXZ    LDA =XZ
   859  FCF2  D0 46               BNE MVTR
   860  FCF4  A9 03       MVXM    LDA =XM
   861  FCF6  D0 42               BNE MVTR
   862  FCF8  A9 04       MVXN    LDA =XN
   863  FCFA  D0 3E               BNE MVTR
   864  FCFC  A9 10       MVYX    LDA =YX
   865  FCFE  D0 3A               BNE MVTR
   866  FD00  A9 12       MVYZ    LDA =YZ
   867  FD02  D0 36               BNE MVTR
   868  FD04  A9 13       MVYM    LDA =YM
   869  FD06  D0 32               BNE MVTR
   870  FD08  A9 14       MVYN    LDA =YN
   871  FD0A  D0 2E               BNE MVTR
   872  FD0C  A9 20       MVZX    LDA =ZX
   873  FD0E  D0 2A               BNE MVTR
   874  FD10  A9 21       MVZY    LDA =ZY
   875  FD12  D0 26               BNE MVTR
   876  FD14  A9 23       MVZM    LDA =ZM
   877  FD16  D0 22               BNE MVTR
   878  FD18  A9 24       MVZN    LDA =ZN
   879  FD1A  D0 1E               BNE MVTR
   880  FD1C  A9 30       MVMX    LDA =MX
   881  FD1E  D0 1A               BNE MVTR
   882  FD20  A9 31       MVMY    LDA =MY
   883  FD22  D0 16               BNE MVTR
   884  FD24  A9 32       MVMZ    LDA =MZ
```

```
CARD =  LOC    CODE        CARD
   885  FD26  DJ 12            BNE MVTR
   886  FD28  A9 34      MVMN  LDA =MN
   887  FD2A  D0 0E            BNE MVTR
   888  FD2C  A9 40      MVNX  LDA =NX
   889  FD2E  D0 0A            BNE MVTR
   890  FD30  A9 41      MVNY  LDA =NY
   891  FD32  D0 06            BNE MVTR
   892  FD34  A9 42      MVNZ  LDA =NZ
   893  FD36  D0 02            BNE MVTR
   894  FD38  A9 43      MVNM  LDA =NM
   895  FD3A  48         MVTR  PHA
   896  FD3B  A2 0B            LDX =11
   897  FD3D  BD 65 FD   MVTRO LDA MOVR,X
   898  FD40  9D 8F 02         STA RAMCOD,X
   899  FD43  CA               DEX
   900  FD44  10 F7            BPL MVTRO
   901  FD46  68               PLA
   902  FD47  48               PHA
   903  FD48  29 0F            AND =$0F
   904  FD4A  AA               TAX
   905  FD4B  BD 60 FD         LDA TAB,X
   906  FD4E  8D 95 02         STA RAMB
   907  FD51  68               PLA
   908  FD52  4A               LSR A
   909  FD53  4A               LSR A
   910  FD54  4A               LSR A
   911  FD55  4A               LSR A
   912  FD56  AA               TAX
   913  FD57  BD 60 FD         LDA TAB,X
   914  FD5A  8D 92 02         STA RAMA
   915  FD5D  4C 8F 02         JMP RAMCOD
   916  FD60  35         TAB   .BYTE $35,$47,$59,$6B,$7D
   916  FD61  47
   916  FD62  59
   916  FD63  6B
   916  FD64  7D
   917  FD65  A2 11      MOVR  LDX =LEN
   918  FD67  BD 35 02   MOVRO LDA RX,X
   919  FD6A  9D 47 02         STA RY,X
   920  FD6D  CA               DEX
   921  FD6E  10 F7            BPL MOVRO
   922  FD70  60               RTS
   923
   924                    SET RX EQUAL TO ZERO.
   925
   926  FD71  A2 11      CLRX  LDX =LEN
   927  FD73  A9 00            LDA =0
   928  FD75  9D 35 02   CLRXO STA RX,X
   929  FD78  CA               DEX
   930  FD79  10 FA            BPL CLRXO
   931  FD7B  60               RTS
   932
```

```
CARD = LOC      CODE        CARD
   933                      SET RY EQUAL TO ZERO.
   934
   935  FD7C  A2 11      CLRY    LDX =LEN
   936  FD7E  A9 00              LDA =0
   937  FD80  9D 47 02   CLRYO   STA RY,X
   938  FD83  CA                 DEX
   939  FD84  10 FA              BPL CLRYO
   940  FD86  60                 RTS
   941
   942                      SET RZ EQUAL TO ZERO.
   943
   944  FD87  A2 11      CLRZ    LDX =LEN
   945  FD89  A9 00              LDA =0
   946  FD8B  9D 59 02   CLRZO   STA RZ,X
   947  FD8E  CA                 DEX
   948  FD8F  10 FA              BPL CLRZO
   949  FD91  60                 RTS
   950
   951                      THIS ROUTINE IS USED TO LOOK UP
   952                      THE COEFFICIENTS OF THE POLY-
   953                      NOMIALS USED IN THE APPROXIMATIONS
   954                      OF THE TRANSCENDENTAL FUNCTIONS.
   955
   956  FD92  20 7C FD   LOOKUP  JSR CLRY
   957  FD95  A2 00              LDX =0
   958  FD97  A4 01              LDY NKON
   959  FD99  B1 0E              LDA (KON),Y
   960  FD9B  8D 47 02           STA SY
   961  FD9E  C8         LKP0    INY
   962  FD9F  B1 0E              LDA (KON),Y
   963  FDA1  C9 F0              CMP =$F0
   964  FDA3  B0 13              BCS LKP1
   965  FDA5  48                 PHA
   966  FDA6  29 0F              AND =$0F
   967  FDA8  9D 49 02           STA RY+2,X
   968  FDAB  68                 PLA
   969  FDAC  4A                 LSR A
   970  FDAD  4A                 LSR A
   971  FDAE  4A                 LSR A
   972  FDAF  4A                 LSR A
   973  FDB0  9D 48 02           STA RY+1,X
   974  FDB3  E8                 INX
   975  FDB4  E8                 INX
   976  FDB5  4C 9E FD           JMP LKP0
   977  FDB8  29 0F      LKP1    AND =$0F
   978  FDBA  8D 58 02           STA EY
   979  FDBD  C8                 INY
   980  FDBE  84 01              STY NKON
   981  FDC0  60                 RTS
   982
   983                      THIS ROUTINE EVALUATES POLYNOMIALS
   984                      BY MEANS OF THE NESTED MULTIPLICATION
```

```
CARD = LOC      CODE        CARD
   985                      ALGORITHM.
   986
   987  FDC1  20 14 FD  POLY    JSR MVZM
   988  FDC4  20 0C FD          JSR MVZX
   989  FDC7  20 92 FD          JSR LOOKUP
   990  FDCA  20 06 F9  POLYO   JSR MLTPLY
   991  FDCD  20 92 FD          JSR LOOKUP
   992  FDD0  20 0C FD          JSR MVZX
   993  FDD3  20 08 F8          JSR ADD
   994  FDD6  20 1C FD          JSR MVMX
   995  FDD9  20 10 FD          JSR MVZY
   996  FDDC  C6 05             DEC DEG
   997  FDDE  10 EA             BPL POLYO
   998  FDE0  60                RTS
   999
  1000                      THIS ROUTINE UNPACKS AN ARGUMENT
  1001                      AND STORES THE RESULT IN RZ.
  1002
  1003  FDE1  A2 00     PGTARG  LDX =0
  1004  FDE3  A0 00             LDY =0
  1005  FDE5  B1 0C             LDA (PTR),Y
  1006  FDE7  8D 59 02          STA SZ
  1007  FDEA  C8        PGTRGO  INY
  1008  FDEB  C4 04             CPY LENGTH
  1009  FDED  F0 15             BEQ PGTRG1
  1010  FDEF  B1 0C             LDA (PTR),Y
  1011  FDF1  48                PHA
  1012  FDF2  29 0F             AND =$0F
  1013  FDF4  9D 5B 02          STA RZ+2,X
  1014  FDF7  68                PLA
  1015  FDF8  4A                LSR A
  1016  FDF9  4A                LSR A
  1017  FDFA  4A                LSR A
  1018  FDFB  4A                LSR A
  1019  FDFC  9D 5A 02          STA RZ+1,X
  1020  FDFF  E8                INX
  1021  FE00  E8                INX
  1022  FE01  4C EA FD          JMP PGTRGO
  1023  FE04  B1 0C     PGTRG1  LDA (PTR),Y
  1024  FE06  8D 6A 02          STA EZ
  1025  FE09  60                RTS
  1026
  1027                      THIS ROUTINE UNPACKS AN ARGUMENT
  1028                      LOCATED AT (ARGXL,ARGXH) AND STORES
  1029                      THE RESULTS IN RZ AND RX.
  1030
  1031  FE0A  A5 06     PLOADX  LDA ARGXL
  1032  FE0C  85 0C             STA PTR
  1033  FE0E  A5 07             LDA ARGXH
  1034  FE10  85 0D             STA PTR+1
  1035  FE12  A5 10             LDA PREC
  1036  FE14  4A                LSR A
```

```
CARD =  LOC     CODE        CARD
 1037   FE15  69 01            ADC =1
 1038   FE17  85 04            STA LENGTH
 1039   FE19  20 87 FD         JSR CLRZ
 1040   FE1C  20 E1 FD         JSR PGTARG
 1041   FE1F  20 0C FD         JSR MVZX
 1042   FE22  60               RTS
 1043
 1044                    THIS ROUTINE UNPACKS AN ARGUMENT
 1045                    LOCATED AT (ARGYL,ARGYH) AND STORES
 1046                    THE RESULT IN RY AND RZ.
 1047
 1048   FE23  A5 08      PLOADY LDA ARGYL
 1049   FE25  85 0C             STA PTR
 1050   FE27  A5 09             LDA ARGYH
 1051   FE29  85 0D             STA PTR+1
 1052   FE2B  A5 10             LDA PREC
 1053   FE2D  4A                LSR A
 1054   FE2E  69 01             ADC =1
 1055   FE30  85 04             STA LENGTH
 1056   FE32  20 87 FD          JSR CLRZ
 1057   FE35  20 E1 FD          JSR PGTARG
 1058   FE38  20 10 FD          JSR MVZY
 1059   FE3B  60                RTS
 1060
 1061                    THIS ROUTINE PACKS THE CONTENTS
 1062                    OF RZ INTO THE LOCATIONS STARTING
 1063                    WITH ADDRESS (RES,RES+1).
 1064
 1065   FE3C  A2 00      PSTRES LDX =0
 1066   FE3E  A0 00             LDY =0
 1067   FE40  AD 59 02          LDA SZ
 1068   FE43  91 0A             STA (RES),Y
 1069   FE45  C8                INY
 1070   FE46  BD 5A 02   PTRES  LDA RZ+1,X
 1071   FE49  0A                ASL A
 1072   FE4A  0A                ASL A
 1073   FE4B  0A                ASL A
 1074   FE4C  0A                ASL A
 1075   FE4D  1D 5B 02          ORA RZ+2,X
 1076   FE50  91 0A             STA (RES),Y
 1077   FE52  C8                INY
 1078   FE53  E8                INX
 1079   FE54  E8                INX
 1080   FE55  E4 10             CPX PREC
 1081   FE57  90 ED             BCC PTRES
 1082   FE59  AD 6A 02          LDA EZ
 1083   FE5C  91 0A             STA (RES),Y
 1084   FE5E  60                RTS
 1085
 1086                    THIS ROUTINE CONVERTS AN ARGUMENT
 1087                    FROM ASCII FORMAT TO COMPUTATIONAL
 1088                    FORMAT AND STORES THE RESULT IN RZ.
```

```
CARD = LOC      CODE        CARD
 1089
 1090   FE5F   A0 00      UGTARG LDY =0
 1091   FE61   B1 0C             LDA (PTR),Y
 1092   FE63   8D 59 02          STA SZ
 1093   FE66   C8         UGTAR0 INY
 1094   FE67   C4 04             CPY LENGTH
 1095   FE69   F0 0A             BEQ UGTAR1
 1096   FE6B   B1 0C             LDA (PTR),Y
 1097   FE6D   29 0F             AND =$0F
 1098   FE6F   99 59 02          STA RZ,Y
 1099   FE72   4C 66 FE          JMP UGTAR0
 1100   FE75   B1 0C      UGTAR1 LDA (PTR),Y
 1101   FE77   0A                ASL A
 1102   FE78   0A                ASL A
 1103   FE79   0A                ASL A
 1104   FE7A   0A                ASL A
 1105   FE7B   8D 6A 02          STA EZ
 1106   FE7E   C8                INY
 1107   FE7F   B1 0C             LDA (PTR),Y
 1108   FE81   29 0F             AND =$0F
 1109   FE83   0D 6A 02          ORA EZ
 1110   FE86   8D 6A 02          STA EZ
 1111   FE89   60                RTS
 1112
 1113                    THIS ROUTINE CONVERTS AN ARGUMENT
 1114                    FROM ASCII FORMAT TO COMP. FORMAT.
 1115                    THE ADDRESS OF THE ARG. IS FOUND IN
 1116                    (ARGXL,ARGXH) AND THE RESULT IS STORED
 1117                    IN RZ AND RX.
 1118
 1119   FE8A   A5 06      ULOADX LDA ARGXL
 1120   FE8C   85 0C             STA PTR
 1121   FE8E   A5 07             LDA ARGXH
 1122   FE90   85 0D             STA PTR+1
 1123   FE92   A5 10             LDA PREC
 1124   FE94   85 04             STA LENGTH
 1125   FE96   E6 04             INC LENGTH
 1126   FE98   20 87 FD          JSR CLRZ
 1127   FE9B   20 5F FE          JSR UGTARG
 1128   FE9E   20 0C FD          JSR MVZX
 1129   FEA1   60                RTS
 1130
 1131                    THIS ROUTINE CONVERTS AN ARGUMENT
 1132                    FROM ASCII FORMAT TO COMP. FORMAT.
 1133                    THE ADDRESS OF THE ARG. IS FOUND IN
 1134                    (ARGYL,ARGYH) AND THE RESULT IS
 1135                    STORED IN RZ AND RY.
 1136
 1137   FEA2   A5 08      ULOADY LDA ARGYL
 1138   FEA4   85 0C             STA PTR
 1139   FEA6   A5 09             LDA ARGYH
 1140   FEA8   85 0D             STA PTR+1
```

```
 1141  FEAA  A5 10        LDA PREC
 1142  FEAC  85 04        STA LENGTH
 1143  FEAE  E6 04        INC LENGTH
 1144  FEB0  20 87 FD     JSR CLRZ
 1145  FEB3  20 5F FE     JSR UGTARG
 1146  FEB6  20 10 FD     JSR MVZY
 1147  FEB9  60           RTS
 1148
 1149                     THIS ROUTINE CONVERTS THE CONTENTS
 1150                     OF RZ TO ASCII FORMAT WHILE MOVING
 1151                     THEM TO THE ADDRESS SPECIFIED BY
 1152                     (RES,RES+1).
 1153
 1154  FEBA  A0 00     USTRES LDY =0
 1155  FEBC  AD 59 02         LDA SZ
 1156  FEBF  91 0A            STA (RES),Y
 1157  FEC1  C8        USTRS0 INY
 1158  FEC2  C4 10            CPY PREC
 1159  FEC4  F0 02            BEQ USTRS1
 1160  FEC6  B0 09            BCS USTRS2
 1161  FEC8  B9 59 02  USTRS1 LDA SZ,Y
 1162  FECB  09 30            ORA =$30
 1163  FECD  91 0A            STA (RES),Y
 1164  FECF  D0 F0            BNE USTRS0
 1165  FED1  C8        USTRS2 INY
 1166  FED2  AD 6A 02         LDA EZ
 1167  FED5  4A               LSR A
 1168  FED6  4A               LSR A
 1169  FED7  4A               LSR A
 1170  FED8  4A               LSR A
 1171  FED9  09 30            ORA =$30
 1172  FEDB  91 0A            STA (RES),Y
 1173  FEDD  C8               INY
 1174  FEDE  AD 6A 02         LDA EZ
 1175  FEE1  29 0F            AND =$0F
 1176  FEE3  09 30            ORA =$30
 1177  FEE5  91 0A            STA (RES),Y
 1178  FEE7  60               RTS
 1179
 1180                     THIS ROUTINE COMPUTES THE
 1181                     INTERNAL PRECISION N FROM
 1182                     PREC AND EXTRA. THE ADD IS
 1183                     A BINARY ADD (UNSIGNED).
 1184
 1185  FEE8  18        IPREC  CLC
 1186  FEE9  A5 10            LDA PREC
 1187  FEEB  65 11            ADC EXTRA
 1188  FEED  85 00            STA N
 1189  FEEF  60               RTS
 1190
 1191                     SAVE THE PROCESSOR INDEX REGISTERS.
 1192
```

```
CARD =  LOC     CODE        CARD
 1193   FEF0   86 15       SAVXY   STX TMPX
 1194   FEF2   84 16               STY TMPY
 1195   FEF4   60                  RTS
 1196
 1197                        RECALL THE PROCESSOR INDEX REGISTERS.
 1198
 1199   FEF5   A6 15       RCLXY   LDX TMPX
 1200   FEF7   A4 16               LDY TMPY
 1201   FEF9   60                  RTS
 1202   FEFA   07 FF       KADDR   .WORD KONST
 1203   FEFC   AD FA FE    SETKON  LDA KADDR
 1204   FEFF   85 0E               STA KON
 1205   FF01   AD FB FE            LDA KADDR+1
 1206   FF04   85 0F               STA KONH
 1207   FF06   60                  RTS
 1208
 1209                        THESE ARE THE COEFFICIENTS USED
 1210                        IN THE EVALUATION OF THE TRANSCENDENTAL
 1211                        FUNCTIONS.
 1212
 1213   FF07   40          KONST   .BYTE $40,$18,$20,$91,$29,$97,$F1
 1213   FF08   18
 1213   FF09   20
 1213   FF0A   91
 1213   FF0B   29
 1213   FF0C   97
 1213   FF0D   F1
 1214   FF0E   40                  .BYTE $40,$55,$34,$27,$38,$70,$F2
 1214   FF0F   55
 1214   FF10   34
 1214   FF11   27
 1214   FF12   38
 1214   FF13   70
 1214   FF14   F2
 1215   FF15   40                  .BYTE $40,$13,$13,$69,$01,$12,$10,$F1
 1215   FF16   13
 1215   FF17   13
 1215   FF18   69
 1215   FF19   01
 1215   FF1A   12
 1215   FF1B   10
 1215   FF1C   F1
 1216   FF1D   40                  .BYTE $40,$17,$31,$09,$55,$17,$F1
 1216   FF1E   17
 1216   FF1F   31
 1216   FF20   09
 1216   FF21   55
 1216   FF22   17
 1216   FF23   F1
 1217   FF24   40                  .BYTE $40,$28,$95,$51,$13,$02,$67,$F1
 1217   FF25   28
 1217   FF26   95
```

```
CARD =  LOC    CODE        CARD
 1217   FF27   51
 1217   FF28   13
 1217   FF29   02
 1217   FF2A   67
 1217   FF2B   F1
 1218   FF2C   40                .BYTE $40,$86,$85,$88,$74,$83,$40,$50,$F1
 1218   FF2D   86
 1218   FF2E   85
 1218   FF2F   88
 1218   FF30   74
 1218   FF31   83
 1218   FF32   40
 1218   FF33   50
 1218   FF34   F1
 1219   FF35   40                .BYTE $40,$93,$26,$42,$67,$F4
 1219   FF36   93
 1219   FF37   26
 1219   FF38   42
 1219   FF39   67
 1219   FF3A   F4
 1220   FF3B   40                .BYTE $40,$25,$54,$91,$79,$60,$F3
 1220   FF3C   25
 1220   FF3D   54
 1220   FF3E   91
 1220   FF3F   79
 1220   FF40   60
 1220   FF41   F3
 1221   FF42   40                .BYTE $40,$17,$42,$11,$19,$88,$F2
 1221   FF43   17
 1221   FF44   42
 1221   FF45   11
 1221   FF46   19
 1221   FF47   88
 1221   FF48   F2
 1222   FF49   40                .BYTE $40,$72,$95,$17,$36,$66,$F2
 1222   FF4A   72
 1222   FF4B   95
 1222   FF4C   17
 1222   FF4D   36
 1222   FF4E   66
 1222   FF4F   F2
 1223   FF50   40                .BYTE $40,$25,$43,$93,$57,$48,$40,$F1
 1223   FF51   25
 1223   FF52   43
 1223   FF53   93
 1223   FF54   57
 1223   FF55   48
 1223   FF56   40
 1223   FF57   F1
 1224   FF58   40                .BYTE $40,$66,$27,$30,$88,$42,$90,$F1
 1224   FF59   66
 1224   FF5A   27
```

```
CARD =  LOC    CODE        CARD
  1224  FF5B   30
  1224  FF5C   88
  1224  FF5D   42
  1224  FF5E   90
  1224  FF5F   F1
  1225  FF60   00                .BYTE $00,$11,$51,$29,$27,$76,$03,$F0
  1225  FF61   11
  1225  FF62   51
  1225  FF63   29
  1225  FF64   27
  1225  FF65   76
  1225  FF66   03
  1225  FF67   F0
  1226  FF68   00                .BYTE $00,$10,$F0
  1226  FF69   10
  1226  FF6A   F0
  1227  FF6B   40                .BYTE $40,$41,$09,$74,$19,$48,$F4
  1227  FF6C   41
  1227  FF6D   09
  1227  FF6E   74
  1227  FF6F   19
  1227  FF70   48
  1227  FF71   F4
  1228  FF72   40                .BYTE $40,$20,$31,$17,$10,$84,$F4
  1228  FF73   20
  1228  FF74   31
  1228  FF75   17
  1228  FF76   10
  1228  FF77   84
  1228  FF78   F4
  1229  FF79   40                .BYTE $40,$27,$97,$43,$35,$03,$70,$F3
  1229  FF7A   27
  1229  FF7B   97
  1229  FF7C   43
  1229  FF7D   35
  1229  FF7E   03
  1229  FF7F   70
  1229  FF80   F3
  1230  FF81   40                .BYTE $40,$98,$34,$59,$45,$39,$30,$F3
  1230  FF82   98
  1230  FF83   34
  1230  FF84   59
  1230  FF85   45
  1230  FF86   39
  1230  FF87   30
  1230  FF88   F3
  1231  FF89   40                .BYTE $40,$39,$86,$59,$10,$47,$05,$F2
  1231  FF8A   39
  1231  FF8B   86
  1231  FF8C   59
  1231  FF8D   10
  1231  FF8E   47
```

```
CARD =  LOC     CODE   CARD
 1231   FF8F   05
 1231   FF90   F2
 1232   FF91   40            .BYTE $40,$16,$14,$89,$77,$76,$17,$40,$F1
 1232   FF92   16
 1232   FF93   14
 1232   FF94   89
 1232   FF95   77
 1232   FF96   76
 1232   FF97   17
 1232   FF98   40
 1232   FF99   F1
 1233   FF9A   40            .BYTE $40,$78,$53,$98,$17,$62,$29,$10,$F1
 1233   FF9B   78
 1233   FF9C   53
 1233   FF9D   98
 1233   FF9E   17
 1233   FF9F   62
 1233   FFA0   29
 1233   FFA1   10
 1233   FFA2   F1
 1234   FFA3   40            .BYTE $40,$28,$49,$88,$96,$20,$80,$F3
 1234   FFA4   28
 1234   FFA5   49
 1234   FFA6   88
 1234   FFA7   96
 1234   FFA8   20
 1234   FFA9   80
 1234   FFAA   F3
 1235   FFAB   C0            .BYTE $C0,$16,$06,$86,$28,$96,$04,$F2
 1235   FFAC   16
 1235   FFAD   06
 1235   FFAE   86
 1235   FFAF   28
 1235   FFB0   96
 1235   FFB1   04
 1235   FFB2   F2
 1236   FFB3   40            .BYTE $40,$42,$69,$15,$19,$27,$11,$F2
 1236   FFB4   42
 1236   FFB5   69
 1236   FFB6   15
 1236   FFB7   19
 1236   FFB8   27
 1236   FFB9   11
 1236   FFBA   F2
 1237   FFBB   C0            .BYTE $C0,$75,$04,$29,$45,$38,$89,$F2
 1237   FFBC   75
 1237   FFBD   04
 1237   FFBE   29
 1237   FFBF   45
 1237   FFC0   38
 1237   FFC1   89
 1237   FFC2   F2
```

```
CARD =  LOC     CODE        CARD
 1238   FFC3   40                  .BYTE $40,$10,$64,$09,$34,$02,$53,$F1
 1238   FFC4   10
 1238   FFC5   64
 1238   FFC6   09
 1238   FFC7   34
 1238   FFC8   02
 1238   FFC9   53
 1238   FFCA   F1
 1239   FFCB   C0                  .BYTE $C0,$14,$20,$36,$44,$46,$65,$20,$F1
 1239   FFCC   14
 1239   FFCD   20
 1239   FFCE   36
 1239   FFCF   44
 1239   FFD0   46
 1239   FFD1   65
 1239   FFD2   20
 1239   FFD3   F1
 1240   FFD4   40                  .BYTE $40,$19,$99,$26,$19,$39,$16,$60,$F1
 1240   FFD5   19
 1240   FFD6   99
 1240   FFD7   26
 1240   FFD8   19
 1240   FFD9   39
 1240   FFDA   16
 1240   FFDB   60
 1240   FFDC   F1
 1241   FFDD   C0                  .BYTE $C0,$33,$33,$30,$73,$34,$50,$50,$F1
 1241   FFDE   33
 1241   FFDF   33
 1241   FFE0   30
 1241   FFE1   73
 1241   FFE2   34
 1241   FFE3   50
 1241   FFE4   50
 1241   FFE5   F1
 1242   FFE6   40                  .BYTE $40,$99,$99,$99,$98,$47,$65,$70,$F1
 1242   FFE7   99
 1242   FFE8   99
 1242   FFE9   99
 1242   FFEA   98
 1242   FFEB   47
 1242   FFEC   65
 1242   FFED   70
 1242   FFEE   F1
 1243                       .END

END OF MOS/TECHNOLOGY 650X ASSEMBLY VERSION 4
NUMBER OF ERRORS =    0,  NUMBER OF WARNINGS =    0
```

SYMBOL TABLE

SYMBOL	VALUE	LINE DEFINED	CROSS-REFERENCES
ADD	F808	74	510 542 993
ADD1	F822	98	187
ADD10	F88B	200	196
ADD110	F887	163	149
ADD12	F88U	157	155
ADD120	F87U	156	141 152 165
ADU121	F883	161	217
ADU13	F8C4	206	136
ADD15	F8D8	216	229 241 247
ADU17	F8E0	223	****
ADD18	F8F0	233	215
ADD19	F8F3	234	****
ADD2	F827	100	89
ADD20	F8F4	239	226
ADD3	F82A	101	91 99 188
ADU31	F832	105	103
ADU4	F83B	109	199 202
ADD5	F84D	129	108 116
ADD6	F892	174	93
ADD7	F899	182	186
ADD8	F89C	183	176
AUU81	F8A9	188	185
ADU9	F8AC	194	104
AMB	FC73	769	774
APB	FC5F	756	762
ARGXH	0007	35	1033 1121
ARGXL	0006	34	1031 1119
ARGYH	0009	37	1050 1139
ARGYL	0008	36	1048 1137
ATANX	FB78	601	****
AZTST	FC99	795	214
AZTSTC	FC9C	797	800
AZU	FBBC	641	643
CHOP	FB36	558	591
CHOPO	FB3A	560	562
CLEAR	FBB8	639	81 255 408
CLRX	FD71	926	****
CLRXC	FD75	928	930
CLRY	FD7C	935	512 533 538 956
CLRYC	FD80	937	939
CLRZ	FD87	944	233 272 352 499 1039 1056 1126 1144
CLRZC	FD8B	946	948
CNT	0003	30	109 257 453 456 474 477 482 653 656 660 665 666 667 668 669 670 671 677 686
CNTA	0004	32	175 310 313 368 780 790
COMPXY	FC82	779	174 420
COM1	FC8A	784	787
COM2	FC96	790	788
DECHEX	FBC3	650	114
DEG	0005	33	551 574 593 608 996
DHCNVE	FBED	673	****
DHCNV1	FBC7	653	658

SYMBOL	VALUE	LINE	DEFINED	CROSS-REFERENCES								
DHCNV2	FBD3	659	655									
DHCNV3	FBDE	665	662									
DIV	FA75	471	424									
DIVEXT	F9DF	368	334									
DIVIDE	FA16	394	507	545								
DIV0	FA79	473	487	488								
DIV1	FA7D	475	478									
DIV2	FA86	479	476									
DIV6	FA30	429	****									
DIV7	FA47	438	436									
DVEXT0	F9E3	370	384									
DVEXT1	F9F0	376	380									
DVEXT2	F9F6	378	373									
DVEXT3	F9FD	381	378									
EX	0246	55	105	129	146	183	197	201	207	300	362	
EXTRA	0011	43	1187									
EY	0258	58	107	184	198	200	303	358	361	518	978	
EZ	026A	61	130	151	164	208	225	228	240	243	323	338
			344	363	371	376	388	843	1024	1082	1105	1109
			1110	1166	1174							
INFIN	FCD2	836	169	350								
INF0	FCD7	839	841									
IPREC	FEE8	1185	****									
J	0002	29	451	452	453	461	472	481	484	485		
KADDR	FEFA	1202	1203	1205								
KON	000E	40	959	962	1204							
KONH	000F	41	1206									
KONST	FF07	1213	1202									
LEN	0011	6	50	51	52	54	57	60	62	63	559	639
			824	917	926	935	944					
LENGTH	0004	31	1008	1038	1055	1094	1124	1125	1142	1143		
LGND0	FB30	556	579									
LKP0	FD9E	961	976									
LKP1	FDB8	977	964									
LOG	FAE7	529	****									
LOGEND	FB28	553	595	610								
LOOKUP	FD92	956	989	991								
LSRA	FB91	614	222	437	463	480						
LSRA0	FB93	615	619	620								
MCOV1	F958	314	320	322								
MDUV2	F960	317	307									
MD1	F93F	302	****									
MD10	F9C3	357	316									
MD100	F95D	316	301									
MD11	F9D9	366	324									
MD2	F970	323	304	315	359	390						
MD22	FA13	390	387									
MD3	F97A	327	365	367								
MD4	F97D	328	****									
MD5	F986	332	356									
MD51	F9A5	345	336	369								
MD59	F9A8	346	305									
MD6	F9AE	348	343	375								
MD61	F9B3	350	311	318	399							
MD7	F9B7	352	309	349	404							
MD8	F9BB	354	329									
MD9	FA08	385	340									

SYMBOL	VALUE	LINE	DEFINED	CROSS-REFERENCES
MLT	FA53	450	290	
MLTPLY	F90B	251	519	549 557 590 606 990
MLTO	FA59	453	462	
MLT1	FA6U	456	459	
MLT2	FA6A	460	457	
MN	0034	22	886	
MOVR	FD65	917	897	
MOVRO	FD67	918	921	
MULT1	F91F	272	263	
MULT2	F923	277	345	377 389
MULT3	F927	282	268	
MULT4	F930	296	438	
MVMN	FD28	886	****	
MVMX	FD1C	880	509	994
MVMY	FD20	882	505	
MVMZ	FD24	884	****	
MVNM	FD38	894	****	
MVNX	FD2C	888	506	537 544
MVNY	FD30	890	555	
MVNZ	FD34	892	****	
MVTR	FD3A	895	857	859 861 863 865 867 869 871 873 875
			877	879 881 883 885 887 889 891 893
MVTRO	FD3D	897	900	
MVXM	FCF4	860	****	
MVXN	FCF8	862	498	532 588 604
MVXY	FCEC	856	589	605
MVXZ	FCFJ	858	572	
MVYM	FD04	868	****	
MVYN	FD08	870	****	
MVYX	FCFC	864	****	
MVYZ	FD00	866	****	
MVZM	FD14	876	504	520 987
MVZN	FD18	878	541	546
MVZX	FD0C	872	511	547 556 988 992 1041 1128
MVZY	FD10	874	508	543 548 578 995 1058 1146
MX	0030	19	880	
MY	0031	20	882	
MZ	0032	21	884	
N	0000	27	115	450 486 530 570 586 602 618 627 679
			692	705 715 725 735 744 754 767 781 795
			805	815 836 1188
NKON	0001	28	497	521 553 576 958 980
NM	0043	26	894	
NX	0040	23	888	
NY	0041	24	890	
NZ	0042	25	892	
OVERR	0014	46	847	
PGTARG	FDE1	1003	1040	1057
PGTRGO	FDEA	1007	1022	
PGTRGI	FE04	1023	1009	
PLOADX	FE0A	1031	****	
PLOADY	FE23	1048	****	
POLY	FDC1	987	554	577
POLYO	FDCA	990	997	
PREC	0010	42	1035	1052 1080 1123 1141 1158 1186
PSTRES	FE3C	1065	****	
PTR	0006	39	1005	1010 1023 1032 1034 1049 1051 1091 1096 1100
			1107	1120 1122 1138 1140

SYMBOL	VALUE	LINE	DEFINED	CROSS-REFERENCES
PTRES	FE46	1070	1081	
RA	0200	50	140 216 319 335 435 615 616 622 629 630	
			634 641 718 737 746 756 760 769 772 797	
RAERQ	FC43	735	434	
RAERX	FC29	715	134 412	
RAMA	0292	65	914	
RAMB	0295	66	906	
RAMCOD	028F	64	898 915	
RAMRB	FC70	767	206 475	
RAPRB	FC5C	754	139 458 479	
RARQ	FC45	736	739	
RARXE	FC35	721	****	
RARXQ	FC2C	717	720	
RB	0212	51	680 681 685 693 695 698 708 757 770	
RBERY	FC1C	705	120 282 416	
RBOF	FC0E	695	700	
RBOFE	FC1B	701	678	
RBOFF	FC07	692	128	
RBRY	FC1F	707	710	
RCLXY	FEF5	1199	****	
RES	000A	38	1068 1076 1083 1156 1163 1172 1177	
RM	026B	62	****	
RN	027D	63	****	
RQ	0224	52	454 483 728 736	
RQERX	FC36	725	286	
RQRX	FC39	727	730	
RSBC	FBF4	680	683	
RSBCNT	FBEE	677	124 687	
RSRA	FBA6	627	142 321 337 460	
RSRAC	FBA9	629	632	
RX	0235	53	717 727 784 806 825 829 918 928	
RY	0247	56	514 516 535 540 707 785 816 826 827 919	
			937 967 973	
RZ	0259	59	501 503 560 747 839 946 1013 1019 1070 1075	
			1098 1161	
RZERA	FC4F	744	161 277	
RZRAE	FC5B	750	****	
RZRAQ	FC52	746	749	
SAVXY	FEF0	1193	****	
SETKON	FEFC	1203	531 571 587 603	
SQRT	FA3E	496	****	
SQRTQ	FAB5	505	522	
SUB	F800	71	536	
SX	0235	54	74 98 148 154 156 195 209 297 308 317	
			325 346 366	
SY	0247	57	71 73 75 296 364 431 433 439 441 960	
SZ	0259	60	157 210 223 245 246 327 331 332 339 347	
			348 355 370 381 383 845 1006 1067 1092 1155	
TAB	FD60	916	905 913	
TANX	FB5C	585	****	
TEMP	0012	44	76 92 102 135 298 299 328	
TEMP1	0013	45	258 306 333 430	
TENX	FB41	569	****	
TMPX	0015	47	1193 1199	
TMPY	0016	48	1194 1200	
UGTARG	FE5F	1090	1127 1145	
UGTARQ	FE66	1093	1099	
UGTARX	FE75	1100	1095	
ULOADX	FE8A	1119	****	

SYMBOL	VALUE	LINE DEFINED		CROSS-REFERENCES	
ULOADY	FEA2	1137	****		
USTRES	FEBA	1154	****		
USTRS0	FEC1	1157	1164		
USTRS1	FEC8	1161	1159		
USTRS2	FED1	1165	1160		
XM	0003	9	860		
XN	0004	10	862		
XSY	FCBF	824	100	182	
XSY1	FCC1	825	831		
XY	0001	7	856		
XZ	0002	8	858		
XZTST	FCA6	805	85	262	403
XZTST0	FCA8	806	809		
XZTST1	FCB2	811	798	807	817
XZTST2	FCB0	810	801	820	
YM	0013	13	868		
YN	0014	14	870		
YY	0010	11	864		
YZ	0012	12	866		
YZTST	FCB3	815	90	267	398
YZTST0	FCB5	816	819		
ZM	0023	17	876		
ZN	0024	18	878		
ZX	0020	15	872		
ZY	0021	16	874		

INSTRUCTION	COUNT
ADC	13
AND	14
ASL	12
BCC	11
BCS	6
BEQ	24
BIT	13
BMI	5
BNE	32
BPL	26
BRK	0
BVC	6
BVS	6
CLC	5
CLD	1
CLI	0
CLV	0
CMP	6
CPX	2
CPY	3
DEC	6
DEX	30
DEY	0
EOR	5
INC	5
INX	9
INY	10
JMP	28
JSR	114
LDA	159
LDX	31
LDY	7
LSR	18
NOP	0
ORA	9
PHA	4
PHP	0
PLA	4
PLP	0
ROL	0
RTI	0
RTS	48
SBC	11
SEC	10
SED	6
SEI	0
STA	118
STX	2
STY	2
TAX	2
TAY	0
TSX	0
TXA	0
TXS	0
TYA	1

= SYMBOLS = 238 (LIMIT = 800) = BYTES = 2032 (LIMIT = 4096)
= LINES = 1449 (LIMIT = 4000) = XREFS = 743 (LIMIT = 1800)

Notizen

Notizen

Notizen

Notizen

ALLES ÜBER MOS -

MICRO'S!

VOM DATENBLATT
ÜBER HANDBÜCHER
BIS ZUM 4 - FARB - POSTER

MCDS
microcomputer
datensysteme gmbh
luisenplatz 4-d 61 darmstadt ☎ 06151/23959

BESTELLKARTE MICROPROCESSOR-DRUCKSCHRIFTEN ENDUSER 2/78 Gültig ab 1. Dez. 1977

MICROCOMPUTER-HANDBÜCHER (Netto + MWSt. + Versand) Wir bestellen zu Ihren bekannten Bedingungen:

1.10	KIM-1 User Manual	englisch	DM	19,80	Stck:	DM:
1.11	KIM-1 Handbuch	deutsch	DM	19,80	Stck:	DM:
1.20	KIM-2/3/4 User Manual Expansion Modules Memory/Errata Sheet/Motherboard	englisch	DM	9,25	Stck:	DM:
1.30	KIMath Subroutines Programming Manual	englisch	DM	15,45	Stck:	DM:
1.40	KIM Text Editor User Manual	englisch	DM	8,90	Stck:	DM:
1.50	KIM Assembler Manual Prelimiary	englisch	DM	10,20	Stck:	DM:
1.60	Programming Manual	englisch	DM	28,60	Stck:	DM:
1.61	Programmierhandbuch	deutsch	DM	28,60	Stck:	DM:
1.62	THE FIRST BOOK OF KIM-1 DAS KIM-REZEPTBUCH	englisch	DM	19,80	Stck:	DM:
1.70	Hardware Manual	englisch	DM	24,90	Stck:	DM:
1.71	Hardware Handbuch	deutsch	DM	24,90	Stck:	DM:
1.80	TIM Terminal Interface Monitor Manual	englisch	DM	12,90	Stck:	DM:
1.90	Cross Assembler Manual Preliminary	englisch	DM	12,70	Stck:	DM:
1.99	Bücherliste über weitere ca. 60 Bücher mit Kurzbeschreibung		DM	--,--	Stck:	DM:

MICROPROCESSOR DATENBLÄTTER (Netto + MWSt. + Versand)

2.10	Spectrum of Products	englisch	DM	0,95	Stck:	DM:
2.20	MCS 6 5 0 0 Microprocessors	englisch	DM	0,95	Stck:	DM:
2.30	MCS 6 5 2 0 Microprocessors	englisch	DM	0,95	Stck:	DM:
2.31	MCS 6 5 2 2 Microprocessors	englisch	DM	0,95	Stck:	DM:
2.40	MCS 6 5 3 0 Microprocessors	englisch	DM	0,95	Stck:	DM:
2.60	MCS 6 5 0 X Instruction Set 6502-6515	englisch	DM	1,10	Stck:	DM:
2.80	MCS 6 5 3 2 Microprocessors	englisch	DM	0,95	Stck:	DM:

MICROCOMPUTER-LOSEBLATT-BIBLIOTHEK (Netto + MWSt. + Versand)

3.10	Loseblattsammelordner aus Kunstleder für alle Bücher und Datenblätter		DM	4,95	Stck:	DM:
3.20	MICRO-Info Hardware ca. 1400 Seiten (FMI)	deutsch	DM	42,00*	Stck:	DM:
3.23	MICRO-Info Software ca. 400 Seiten (FMI)	deutsch	DM	38,00*	Stck:	DM:
3.26	Kontinuierliche automatische Ergänzung zur (FMI) 2-monatlich ca. 150 Seiten,10 DM/S	deutsch	DM	15,00	Abo:	DM:

*) Gilt nur in Verbindung mit Abo, sonst 84,—/76,— DM.

MICRO-COMPUTER-POSTER UND PROSPEKTE (Netto + MWSt. + Versand)

4.10	KIM-1 Microcomputer Module, 4-Farb-Poster Schaltschema ☐ = ungefaltet ☐ gefaltet A 4		DM	2,95	Stck:	DM:
9.00	MCDS-HARDWARE + SOFTWARE PRODUKTE Schnell-INFO		DM	0,80	Stck:	DM:

COMPUTERJOURNALE (Enduserpreise incl. MWSt. + Porto)

11.10	JOCE & N: Europas einziges Computerjournal in **deutsch und englisch** für Hobby, Forschung und Ausbildung	englisch/deutsch Abonnement	DM	4,00 40,00	Stck:	DM:
11.11	KILOBAUD: Das bekannte US-Journal des MICRO + MINICOMPUTERMARKTES	englisch Abonnement	DM	7,50 75,00	Stck:	DM:
11.13	INTERFACE AGE: Ebenso bekanntes wie beliebtes US-Computer- Journal	englisch Abonnement	DM	6,00 60,00	Stck:	DM:
11.15	THE NEW HOBBY COMPUTERS, der BESTE Jahresextrakt	englisch	DM	15,75	Stck:	DM:
11.16	THE HOBBY COMPUTERS ARE HERE, der BESTE Jahresextrakt	englisch	DM	15,75	Stck:	DM:
11.17	„73" MAGAZINE AMATEUR RADIO für Funkamateure in der BRD, 3500 Abos	englisch Abonnement	DM	7,00 70,00	Stck:	DM:

COMUPTER CLUB EUROPE E.V.

21.00	MITGLIEDSCHAFT, kostenfreien Bezug des JOCE & N + europaweite Kommunikation Rabatte auf Soft- und Hardware und vieles mehr.	Beitrag/Jahr Student/Jahr	DM	85,00 45,00	☐ ☐

Alle Preise verstehen sich netto fob Darmstadt.

Nachname: Vorname:

Straße:

Postleitzahl-Landeskennbuchstabe: Ort:

Datum: Unterschrift:

MCDS MICROCOMPUTER Datensysteme GmbH, Luisenplatz 4, D-6100 Darmstadt

MOS TECHNOLOGY, INC.

microcomputer
datensysteme gmbh.

luisenplatz 4.
d 61 darmstadt.
☎ 06151/23959

This was brought to you

from the archives of

http://retro-commodore.eu