MATH PACK UTILITIES

OPERATOR'S MANUAL

TransEra

3707 North Canyon Road • Suite 4
Provo, Utah 84604

Phone 801-224-6550

TABLE OF CONTENTS

GENERAL DESCRIPTION

Introduction

The 650/750-MPU Math Pack Utilities ROM Pack is a collection of math functions that perform a range of useful operations from complex arithmetic to Fourier Transforms. Each of these routines replace several lines of BASIC code and execute much faster than BASIC. They are accessed through the CALL statement in BASIC and are written in assembly language for the greatest possible execution speed.

There are a total of 23 routines in the ROM Pack. Six of the routines perform complex math functions, six perform hyperbolic trigonometric functions, seven perform various array handling functions, and four perform Fourier and Inverse Fourier Transforms.

The routines in this ROM Pack are explained in the following manner. The format of the CALL statement is given and calling parameter is explained. The general operation of the routine is then explained.

In the listing of the calling parameters, the letters O and I following the parameter name indicate whether the parameter is used as an input to the routine, an output from the routine, or both. In general, input parameters may be variables, expressions, or constants. Output parameters must be variables. Error checking is done by the routine to insure correct typing of the calling parameters.

As far as possible, the algorithms used in these routines return the principal solution in cases where more than one solution is possible.

Installation Instructions

The power to the 4050 should be turned off before the ROM Pack is installed. After the power is shut off, the ROM Pack may be inserted into a slot in the firmware backpack or into a slot of a ROM Expander Unit. Press down gently until the edge card connector is seated in the receptacle connector.

Explanation of Calling Parameters

The complex math routines and hyperbolic trig routines in this ROM Pack use complex calling parameter sets. Each complex calling parameter set represents one complex number. They are represented with either a two element array variable or two scalar variables. A single scalar variable may also be used which will mean that the imaginary part of the complex number is to be ignored or discarded. Since these different variable combinations are allowed to represent complex numbers, it is necessary to separate each parameter set with a semicolon. In parameter sets which are inputs to a routine, expressions or constants may be used in place of scalar variables.

Example: `CALL "SINH",A;B`

A and B are two element arrays. B(1) and B(2) are used as the real and imaginary parts of the function argument, and the results are put into A(1) and A(2).

```
CALL "SINH",A;X,Y
```

A is a two element array, and X and Y are scalar variables. X and Y are used as the function argument, and the result is put into A.

```
CALL "SINH",W,X;Z+3
```

W, X, and Z are scalar variables. Z+3 (with 0 as the imaginary part) is used as the function argument, and the result is put into W and X.

```
CALL "SINH",X;1,2
```

X is a scalar variable. 1 and 2 are used as the function argument, and the real result is put into X. The imaginary result is discarded.

COMPLEX MATH ROUTINES

Introduction

There are six complex math routines in the Math Pack Utilities ROM Pack. They are RTP, PTR, MULT, DIV, CPWR, and CLOG. RTP converts from rectangular form to polar form. PTR converts from polar form to rectangular form. MULT multiplies two complex numbers in polar form. DIV divides two complex numbers in polar form. CPWR raises a conplex number to a complex power. CLOG takes the logarithm of a complex number to a complex base.

RTP (Rectangular to Polar)

```
CALL "RTP",A;B

 A:O  OUTPUT PARAMETER SET, POLAR FORM
 B:I  INPUT PARAMETER SET, RECTANGULAR FORM
```

RTP converts a complex number in rectangular form to a complex number in polar form. The polar form representation for A defines the first element or variable as the radius and the second element or variable as the angle from the positive real axis. RTP is implemented from the following algorithm:

```
A(1)=SQR(B(1)^2+B(2)^2)
A(2)=ACS(B(1)/(SQR(B(1)^2+B(2)^2))*SGN(B(2))
```

Example:

```
100 PRINT "ENTER REAL PART: ";
110 INPUT X
120 PRINT "ENTER IMAGINARY PART: ";
130 INPUT Y
140 CALL "RTP",R,A;X,Y
150 PRINT "RADIUS: ";R
160 PRINT " ANGLE: ";A
170 END
```

Output:

```
ENTER REAL PART: 3
ENTER IMAGINARY PART: 4
RADIUS: 5
 ANGLE: 0.927295218002
```

In this example, RTP converts the rectangular form complex number (3,4) to the polar form complex number (5,0.927295218002). In this example, each of the complex parameter sets consists of two scalar variables.

PTR (Polar to Rectangular)

```
CALL "PTR",A;B

 A:O  OUTPUT PARAMETER SET, RECTANGULAR FORM
 B:I  INPUT PARAMETER SET, POLAR FORM
```

RTP converts a complex number in polar form to a complex number in rectangular form. The polar form representation defines the first element or variable as the radius and the second element or variable as the angle from the positive real axis. PTR is implemented from the following algorithm:

```
A(1)=B(1)*COS(B(2))
A(2)=B(1)*SIN(B(2))
```

Example:

```
100 DIM A(2),B(2)
110 PRINT "ENTER RADIUS: ";
120 INPUT B(1)
130 PRINT "ENTER ANGLE: ";
140 INPUT B(2)
150 CALL "PTR",A;B
160 PRINT "REAL PART: ";A(1)
170 PRINT "IMAGINARY PART: ";A(2)
180 END
```

Output:

```
ENTER RADIUS: 5
ENTER ANGLE: 0.927295218002
REAL PART: 3
IMAGINARY PART: 4
```

In this example, PTR converts the polar form complex number (5,0.927295218002) to the rectangular form complex number (3,4). In this example, each of the complex parameter sets consists of a two element array.

MULT (Complex Multiply)

```
CALL "MULT",A;B;C
```

```
 A:O  OUTPUT PARAMETER SET, RECTANGULAR FORM
 B:I  INPUT PARAMETER SET, RECTANGULAR FORM
 C:I  INPUT PARAMETER SET, RECTANGULAR FORM
```

MULT multiplies two complex numbers in rectangular form. It is analogous to the statement A=B*C. It is implemented from the following algorithm:

```
A(1)=B(1)*C(1)-B(2)*C(2)
A(2)=B(1)*C(2)+B(2)*C(1)
```

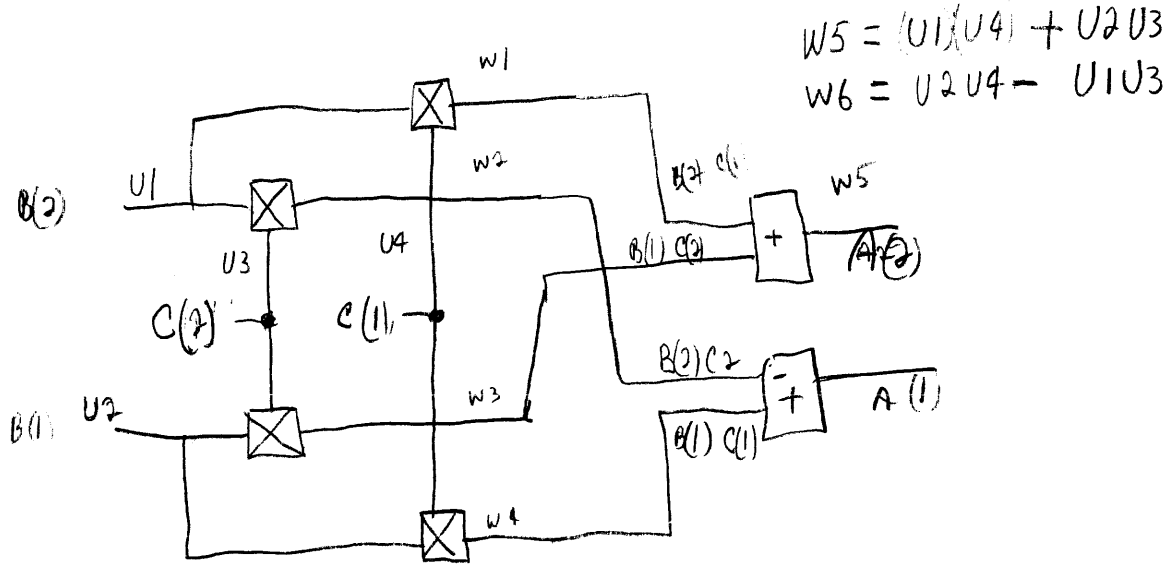
Example:

```
100 DIM A(2),B(2),C(2)
110 PRINT "ENTER B: ";
120 INPUT B
130 PRINT "ENTER C: ";
140 INPUT C
150 CALL "MULT",A;B;C
160 PRINT "PRODUCT: ";A(1);A(2)
170 END
```

Output:

```
ENTER B: 1,2
ENTER C: 3,4
PRODUCT: -5 10
```

In this example, MULT multiplies the complex numbers (1,2) and (3,4) and puts the result in array A.



DIV (Complex Divide)

```
CALL "DIV",A;B;C

 A:O  OUTPUT PARAMETER SET, RECTANGULAR FORM
 B:I  INPUT PARAMETER SET, RECTANGULAR FORM
 C:I  INPUT PARAMETER SET, RECTANGULAR FORM
```

DIV divides two complex numbers in rectangular form. It is analogous to the statement A=B/C. It is implemented from the following algorithm:

```
A(1)=(B(1)*C(1)+B(2)*C(2))/(C(1)^2+C(2)^2)
A(2)=(B(2)*C(1)-B(1)*C(2))/(C(1)^2+C(2)^2)
```

Example:

```
100 DIM A(2),B(2),C(2)
110 PRINT "ENTER B: ";
120 INPUT B
130 PRINT "ENTER C: ";
140 INPUT C
150 CALL "DIV",A;B;C
160 PRINT "RESULT: ";A(1);A(2)
170 END
```

Output:

```
ENTER B: 1,2
ENTER C: 3,4
RESULT: 0.44 0.08
```

In this example, DIV divides the complex numbers (1,2) and (3,4) and puts the result in array A.

CPWR (Complex Power)

```
CALL "CPWR",A;B;C
```

```
 A:O  OUTPUT PARAMETER SET, RECTANGULAR FORM
 B:I  INPUT PARAMETER SET, RECTANGULAR FORM
 C:I  INPUT PARAMETER SET, RECTANGULAR FORM
```

CPWR raises one complex number to the power denoted by another complex number. It is analogous to the statement A=B^C. It is implemented from the following algorithm:

```
R=SQR(B(1)^2+B(2)^2)
S=ACS(B(1)/R)*SGN(B(2))
R=LOG(R)
Z1=C(1)*R-C(2)*S
Z2=C(1)*S+C(2)*R
Z1=EXP(Z1)
A(1)=Z1*COS(Z2)
A(2)=Z1*SIN(Z2)
```

If R is found to be zero, A(1) and A(2) are set to zero.

Example:

```
100 DIM A(2),B(2),C(2)
110 PRINT "ENTER B: ";
120 INPUT B
130 PRINT "ENTER C: ";
140 INPUT C
150 CALL "CPWR",A;B;C
160 PRINT "RESULT: ";A(1);A(2)
170 END
```

Output:

```
ENTER B: 1,2
ENTER C: 3,4
RESULT: 0.129009594074 0.0339240929052
```

In this example, CPWR raises the complex number (1,2) to the (3,4) power and puts the result in array A.

CLOG (Complex Logarithm)

```
CALL "CLOG",A;B;C
```

A:O OUTPUT PARAMETER SET, RECTANGULAR FORM
B:I INPUT PARAMETER SET, RECTANGULAR FORM
C:I INPUT PARAMETER SET, RECTANGULAR FORM

CLOG takes the logarithm of a complex number to a complex base. It is analogous to the statement A=LOG(B)/LOG(C) which takes the logarithm of B base C. It is implemented from the following algorithm:

```
R1=SQR(B(1)^2+B(2)^2)
R2=SQR(C(1)^2+C(2)^2)
S1=ACS(B(1)/R1)*SGN(B(2))
S2=ACS(C(1)/R2)*SGN(C(2))
R1=LOG(R1)
R2=LOG(R2)
Z1=R2^2+S2^2
A(1)=(R1*R2+S1*S2)/Z1
A(2)=(S1*R2-R1*S2)/Z1
```

If either R1 or R2 is found to be zero after the first steps, an error message 23 is generated since the logarithm of zero is undefined. If Z1 is found to be zero, a size error is generated.

Example:

```
100 DIM A(2),B(2),C(2)
110 PRINT "ENTER B: ";
120 INPUT B
130 PRINT "ENTER C: ";
140 INPUT C
150 CALL "CLOG",A;B;C
160 PRINT "RESULT: ";C(1);C(2)
170 END
```

Output:

```
ENTER B: 1,2
ENTER C: 3,4
RESULT: 0.67295265212 0.300181161267
```

In this example, CLOG takes the logarithm of the complex number (1,2) to base (3,4) and puts the result in array A.

HYPERBOLIC TRIG ROUTINES

Introduction

There are six hyperbolic trig routines in the Math Pack Utilities ROM Pack. They are SINH, COSH, TANH, ASINH, ACOSH, and ATANH. SINH, COSH, and TANH compute the hyperbolic sine, cosine, and tangent of a complex or real number, respectively. ASINH, COSH, and ATANH perform the corresponding inverse functions.

Since hyperbolic trig functions are not built into the 4050 BASIC language, these routines may be profitably used with real data. To do this, use a scalar variable, constant, or expression for the input parameter sets and a scalar variable for the output parameter set. These parameters must still be separated by a semicolon.

SINH (Complex Hyperbolic Sine)

```
CALL "SINH",A;B
```

A:O OUTPUT PARAMETER SET, RECTANGULAR FORM
B:I INPUT PARAMETER SET, RECTANGULAR FORM

SINH takes the hyperbolic sine of a complex number represented by parameter set B and puts the result in the variables passed in parameter set A. It is implemented from the following algorithm:

```
Z1=EXP(B(1))
A(1)=0.5*COS(B(2))*(Z1-1/Z1)
A(2)=0.5*SIN(B(2))*(Z1+1/Z1)
```

Example:

```
100 DIM A(2),B(2)
110 PRINT "ENTER B: ";
120 INPUT B
130 CALL "SINH",A;B
140 PRINT "RESULT: ";A(1);A(2)
150 END
```

Output:

```
ENTER B: 1,2
RESULT: -0.489056259041 1.40311925062
```

In this example, SINH takes the hyperbolic sine of the complex number (1,2) and puts the result in array A.

COSH (Complex Hyperbolic Cosine)

```
CALL "COSH",A;B

 A:O  OUTPUT PARAMETER SET, RECTANGULAR FORM
 B:I  INPUT PARAMETER SET, RECTANGULAR FORM
```

COSH takes the hyperbolic cosine of a complex number represented by parameter set B and puts the result in the variables passed in parameter set A. It is implemented from the following algorithm:

```
Z1=EXP(B(1))
A(1)=0.5*COS(B(2))*(Z1+1/Z1)
A(2)=0.5*SIN(B(2))*(Z1-1/Z1)
```

Example:

```
100 DIM A(2),B(2)
110 PRINT "ENTER B: ";
120 INPUT B
130 CALL "COSH",A;B
140 PRINT "RESULT: ";A(1);A(2)
150 END
```

Output:

```
ENTER B: 1,2
RESULT: -0.642148124716 1.06860742138
```

In this example, COSH takes the hyperbolic cosine of the complex number (1,2) and puts the result in array A.

TANH (Complex Hyperbolic Tangent)

```
CALL "TANH",A;B

 A:O  OUTPUT PARAMETER SET, RECTANGULAR FORM
 B:I  INPUT PARAMETER SET, RECTANGULAR FORM
```

TANH takes the hyperbolic tangent of a complex number represented by parameter set B and puts the result in the variables passed in parameter set A. It is implemented from the following algorithm:

```
Z1=EXP(2*B(1))
Z2=1/Z1
Z3=(Z1+Z2)/2+COS(2*B(2))
A(1)=(Z1-Z2)/2/Z3
A(2)=SIN(2*B(2))/Z3
```

If Z3 is zero, the value of the TANH is infinite, so a size error is generated.

Example:

```
100 DIM A(2),B(2)
110 PRINT "ENTER B: ";
120 INPUT B
130 CALL "TANH",A;B
140 PRINT "RESULT: ";A(1);A(2)
150 END
```

Output:

```
ENTER B: 1,2
RESULT: 1.16673625724 -0.243458201186
```

In this example, TANH takes the hyperbolic tangent of the complex number (1,2) and puts the result in array A.

ASINH (Complex Hyperbolic Arc Sine)

```
CALL "ASINH",A;B

 A:O  OUTPUT PARAMETER SET, RECTANGULAR FORM
 B:I  INPUT PARAMETER SET, RECTANGULAR FORM
```

ASINH takes the arc hyperbolic sine of a complex number represented by parameter set B and puts the result in the variables passed in parameter set A. It is implemented from the following algorithm:

```
X1=(1-B(2))^2+B(1)^2
X2=SQR(X1+4*B(2))
X1=SQR(X1)
Z1=0.5*(X1+X2)
Z2=0.5*(X1-X2)
A(2)=-ASN(Z2)
A(1)=LOG(Z1+SQR(Z1^2-1))
```

Example:

```
100 DIM A(2),B(2)
110 PRINT "ENTER B: ";
120 INPUT B
130 CALL "ASINH",A;B
140 PRINT "RESULT: ";A(1);A(2)
150 END
```

Output:

```
ENTER B: 1,2
RESULT: 1.46935174437 1.06344002358
```

In this example, ASINH takes the arc hyperbolic sine of the complex number (1,2) and puts the result in array A.

ACOSH (Complex Hyperbolic Arc Cosine)

```
CALL "ACOSH",A;B

 A:O  OUTPUT PARAMETER SET, RECTANGULAR FORM
 B:I  INPUT PARAMETER SET, RECTANGULAR FORM
```

ACOSH takes the arc hyperbolic cosine of a complex number represented by parameter set B and puts the result in the variables passed in parameter set A. It is implemented from the following algorithm:

```
X1=(B(1)+1)^2+B(2)^2
X2=SQR(X1-4*B(1))
X1=SQR(X1)
Z1=0.5*(X1+X2)
Z2=0.5*(X1-X2)
A(2)=ACS(Z2)
A(1)=LOG(Z1+SQR(Z1^2-1))
```

Example:

```
100 DIM A(2),B(2)
110 PRINT "ENTER B: ";
120 INPUT B
130 CALL "ACOSH",A;B
140 PRINT "RESULT: ";A(1);A(2)
150 END
```

Output:

```
ENTER B: 1,2
RESULT: 1.52857091948 1.1437177404
```

In this example, ACOSH takes the arc hyperbolic cosine of the complex number (1,2) and puts the result in array A.

ATANH (Complex Hyperbolic Arc Tangent)

```
CALL "ATANH",A;B
```

A:O OUTPUT PARAMETER SET, RECTANGULAR FORM
B:I INPUT PARAMETER SET, RECTANGULAR FORM

ATANH takes the arc hyperbolic tangent of a complex number represented by parameter set B and puts the result in the variables passed in parameter set A. It is implemented from the following algorithm:

```
R1=B(1)^2+B(2)^2
A(2)=0.5*ATN(2*B(2)/(1-R1))
A(1)=-0.25*LOG((R1-2*B(1)+1)/(R1+2*B(1)+1))
```

Example:

```
100 DIM A(2),B(2)
110 PRINT "ENTER B: ";
120 INPUT B
130 CALL "ATANH",A;B
140 PRINT "RESULT: ";A(1);A(2)
150 END
```

Output:

```
ENTER B: 1,2
RESULT: 0.17328679514 -0.392699081699
```

In this example, ATANH takes the arc hyperbolic tangent of the complex number (1,2) and puts the result in array A.

ARRAY ROUTINES

Introduction

There are seven array routines in the Math Pack Utilities ROM Pack. They are INTERP, INTEG, DERIV, PACK, UNPACK, ARYSET, and PLOT$. INTERP performs linear interpolation of array data. INTEG performs integration of array data. DERIV performs first or second differentiation of array data. PACK converts array data to packed binary string data for more compact storage or manipulation. UNPACK converts packed binary string data into an array. ARYSET defines an array with a starting value and an incremental value for each successive array element. PLOT$ draws graphs from binary string data.

INTERP (Interpolate)

CALL "INTERP",X,Y,B,C[,C1]

```
 X:I  SOURCE ARRAY, OLD INDEPENDENT VARIABLE
 Y:I  SOURCE ARRAY, OLD DATA
 B:I  SOURCE ARRAY, NEW INDEPENDENT VARIABLE
 C:O  TARGET ARRAY, NEW DATA
C1:I  INITIAL FILL VALUE (OPTIONAL)
```

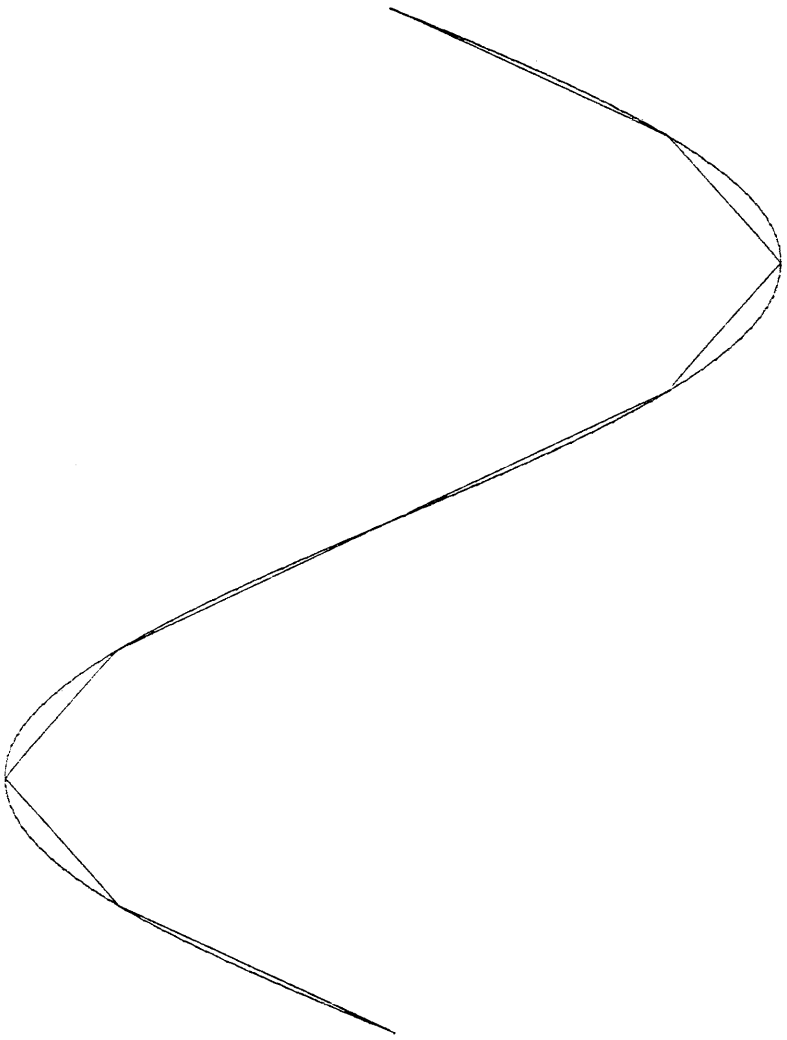
INTERP performs linear interpolation on the original data in array Y with the independent variable in array X onto the new independent variable in array B. The new data is stored in array C. The range of B should be less than or equal to the range of X. C1 is the optional fill value for C and has a default value of zero. INTERP is implemented from the following BASIC program:

```
2480 REM START OF BASIC INTERP
2490 C=C1
2500 J1=1
2510 K=X(N5)>X(1)
2520 FOR I=1 TO N4
2530 L0=B(I)
2540 IF L0<X(K+N5*(1-K)) OR L0>X(N5*K+1-K) THEN 2610
2550 FOR J=J1 TO N5-1
2560 J1=J
2570 IF L0<X((J+1)*K+J*(1-K)) AND L0=>X(J*K+(J+1)*(1-K)) THEN 2590
2580 NEXT J
2590 J=J-(J>N5-1)
2600 C(I)=(Y(J+1)-Y(J))/(X(J+1)-X(J))*(L0-X(J))+Y(J)
2610 NEXT I
2620 REM END OF BASIC INTERP
```

Example:

```
100 N5=100
110 N4=9
120 DIM X(N5),Y(N5),B(N4),C(N4)
130 FOR I=1 TO N5
140 X(I)=I
150 Y(I)=SIN((I-1)*2*PI/(N5-1))
160 NEXT I
170 FOR I=1 TO N4
180 B(I)=1+(I-1)*(N5-1)/(N4-1)
190 NEXT I
200 CALL "INTERP",X,Y,B,C
210 WINDOW 1,N5,-1,1
220 MOVE X(1),Y(1)
230 DRAW X,Y
240 MOVE B(1),C(1)
250 DRAW B,C
260 END
```

In this example, arrays X and Y approximate a sine wave in 100 points. INTERP interpolates this curve into arrays B and C, which then approximate a sine wave in 9 points. This is what the plot looks like:



INTEG (Integrate)

```
CALL "INTEG",X,Y,B
```

```
 X:I  SOURCE ARRAY, DATA
 Y:I  SOURCE ARRAY, INDEPENDENT VARIABLE
 B:O  TARGET ARRAY, INTEGRAL OF DATA
```

INTEG performs integration of the data in array X over the independent variable in array Y. The result for each element is stored in array B. It is implemented from the following BASIC program:

```
2840 REM BASIC INTEG
2850 S=0
2860 K=2*(X(N1)>X(1))-1
2870 K1=K<>1
2880 B(1)=0
2890 B(N1)=0
2900 FOR I=1 TO N1-1
2910 J=I*K+(N1+1)*K1
2920 S=S+0.5*(Y(J)+Y(J+K))*(X(J+K)-X(J))
2930 B(J+K)=S
2940 NEXT I
2950 REM END OF BASIC INTEG
```

~~Example:~~

```
~~100 N1=10~~
110 DIM X(N1),Y(N1),B(N1)
120 FOR I=1 TO N1
130 X(I)=I^2
140 Y(I)=I
150 NEXT I
160 CALL "INTEG",X,Y,B
170 PRINT X,Y,B
180 END
```

Output:

```
 1       4       9       16
 25      36      49      64
 81      100

 1       2       3       4
 5       6       7       8
 9       10

 0       4.5     17      41.5
 82      142.5   227     339.5
 484     664.5
```

In this example, INTEG integrates the data in array X over the independent variable in array Y and stores the result for each point in array B.

DERIV (Derivative)

```
CALL "DERIV",A,Z,A1,N2
```

```
 A:I  SOURCE ARRAY, DATA
 Z:I  SOURCE ARRAY, INDEPENDENT VARIABLE
A1:O  TARGET ARRAY, DERIVATIVE OF DATA
N2:I  ORDER OF DERIVATIVE
```

DERIV performs the derivative of the data in array A with respect to the independent variable in array Z. The result for each element is stored in array A1. N2 determines the order of the derivative taken and should be 1 or 2. The first and last point of a second order derivative are not calculated and are set to zero. DERIV is implemented from the following BASIC program:

```
3120 REM START OF BASIC DERIV
3130 IF N2=2 THEN 3230
3140 FOR I=2 TO I2-1
3150 D1=Z(I)-Z(I-1)
3160 D4=Z(I+1)-Z(I)
3170 D3=1/(D1*D4*(D1+D4))
3180 A1(I)=(D4*D4*(A(I)-A(I-1))+D1*D1*(A(I+1)-A(I)))*D3
3190 NEXT I
3200 A1(1)=(A(2)-A(1))/(Z(2)-Z(1))
3210 A1(I2)=(A(I2)-A(I2-1))/(Z(I2)-Z(I2-1))
3220 GO TO 3320
3230 REM SECOND DERIVATIVE
3240 FOR I=2 TO I2-1
3250 D1=Z(I)-Z(I-1)
3260 D4=Z(I+1)-Z(I)
3270 D3=1/(D1*D4*(D1+D4))
3280 A1(I)=2*(D4*(A(I-1)-A(I))+D1*(A(I+1)-A(I)))*D3
3290 NEXT I
3300 A1(1)=0
3310 A1(I2)=0
3320 REM END OF BASIC DERIV
```

Example:

```
100 I2=10
110 DIM A(I2),Z(I2),A1(I2),A2(I2)
120 FOR I=1 TO I2
130 A(I)=I^2
140 Z(I)=I
150 NEXT I
160 CALL "DERIV",A,Z,A1,1
170 CALL "DERIV",A,Z,A2,2
180 PRINT A,Z,A1,A2
190 END
```

Output:

```
 1        4        9        16
 25       36       49       64
 81       100

 1        2        3        4
 5        6        7        8
 9        10

 3        4        6        8
 10       12       14       16
 18       19

 0        2        2        2
 2        2        2        2
 2        0
```

In this example, DERIV performs the derivative of the data in array A with respect to the independent variable in array Z. The result of the first order derivative is put in array A1, and the result of the second order derivative is put in array A2.

PACK

CALL "PACK",A$,A,N,B

A$:O TARGET STRING
A:I SOURCE ARRAY
N:I NUMBER OF POINTS TO CONVERT
B:I BYTES PER POINT

PACK converts floating point data in an array to packed binary data in a string variable. PACK takes data from array A, converts it, and puts the converted data into string variable A$. N is the number of points to convert. B is the number of bytes of A$ to use for each converted point and may be either 1 or 2. Array A must contain at least N defined elements. String variable A$ must be dimensioned to at least B*N bytes.

PACK rounds each element to be converted to the nearest whole number and packs it into binary format. If B is 1, values of 0 to 255 are valid. If B is 2, values of 0 to 65535 are valid.

Example:

```
100 DIM A(5),A$(5)
110 READ A
120 DATA 48,49,50,51,52
130 CALL "PACK",A$,A,5,1
140 PRINT A$
150 END
```

Output:

```
01234
```

In this example, PACK converts five elements from A to 1-byte binary format and puts the converted data in A$. A$ thus contains 5 characters when pack has finished.

UNPACK

```
CALL "UNPACK",A$,A,N,B

A$:I  SOURCE STRING
 A:O  TARGET ARRAY
 N:I  NUMBER OF POINTS TO DO
 B:I  BYTES PER POINT
```

UNPACK converts packed binary data from string A$ and puts the converted values into floating point array A. It is essentially the reverse of the PACK routine. N is the number of points to convert. B is the number of bytes to take from A$ for each conversion and may be either 1 or 2. Array A must be dimensioned to at least N bytes and string A$ must contain at least N*B characters.

UNPACK converts each binary value to the corresponding integer in floating point format. If B is 1, values from 0 to 255 may be produced. If B is 2, values from 0 to 65535 may be produced.

Example:

```
100 DIM A(5)
110 CALL "UNPACK","01234",A,5,1
120 PRINT A
130 END
```

Output:

```
48      49      50      51
52
```

In this example, UNPACK takes its data from the literal string "01234" and puts the converted data into array A. Five points are converted, using one byte from the source string for each point.

ARYSET (Array Set)

```
CALL "ARYSET",A,M,B

 A:O  TARGET ARRAY
 M:I  VALUE FOR A(1)
 B:I  INCREMENTAL VALUE
```

Aryset defines an array with a starting value and an incremental value for each successive array element. The array must have been previously DIMENSIONed to the desired number of elements. It may be one or two dimensional. The value of M is assigned to the first element in the array. M+B is assigned to the second element, and so on. ARYSET stops when it encounters the dimensioned end of the array.

Example 1:

```
100 DIM A(5)
110 CALL "ARYSET",A,100,10
120 PRINT A
130 END
```

Output:

```
100      110      120      130
140
```

In this example, ARYSET assigns the value 100 to the first element of the array A, 110 to the second element, and so on.

Example 2:

```
100 DIM X(100),Y(100)
110 CALL "ARYSET",X,0,1
120 CALL "ARYSET",Y,0,PI*2/99
130 Y=SIN(Y)
140 WINDOW 0,99,-1,1
150 MOVE X(1),Y(1)
160 DRAW X,Y
170 END
```

In this example, ARYSET assigns values 0 through 99 to elements of X and value 0 through PI*2 to elements of Y. Line 130 computes the sine of each element of Y. Lines 140 through 160 plot the sine wave defined by X and Y to the screen.

PLOT$

CALL "PLOT$",A$,D,I,S,N,B

A$:I STRING TO BE GRAPHED
 D:I DEVICE NUMBER FOR GRAPHING
 I:I INTERVAL
 S:I STARTING POINT
 N:I NUMBER OF POINTS TO PLOT
 B:I NUMBER OF BYTES PER SAMPLE

The PLOT$ routine draws a graph of data in a string. A$ is the string containing the data to be graphed. D is the device to draw to. It must be a legal value. I is the incremental value between plotted points. It should be 1 for unmultiplexed data. S is the point in the string to begin graphing at. N is the Number of points to plot. B is the number of bytes (1 or 2) per sample.

PLOT$ draws a graph of every I`th point starting at number S. It draws until N points have been graphed. The graph is drawn to device D. A$ must be a defined string. All other parameters may be simple variables, expressions, or literal values. Before calling PLOT$, the WINDOW should be set to N in the horizontal direction and the largest expected sample in the vertical direction. PLOT$ may be used to graph the used in the signal processing routines, but the WINDOW statement must be from 0 to 65536, rather than from -32768 to 32768.

Example:

```
100 DIM A$(100),A(50)
110 FOR I=1 TO 50
120 A(I)=32768*(1+SIN(PI*I/25))
130 NEXT I
140 CALL "PACK",A$,A,50,2
150 WINDOW 1,50,0,65535
160 CALL "PLOT$",A$,32,1,1,50,2
170 END
```

In this example, lines 100 through 140 create a string of 50 2-byte words of data. Line 150 sets the plotting window to the correct size. Line 160, PLOT$, plots the data to the screen using an interval of 1 and a starting location of 1. This plots all 50 2-byte words and draws one cycle of a sine wave on the screen.

SIGNAL PROCESSING ROUTINES

Introduction

There are four signal processing routines in the Math Pack Utilities ROM Pack. They are FFT$, IFT$, PACK$, and UNPAK$. FFT$ computes the Fourier Transform of up to 8192 points of packed data. IFT$ computes the Inverse Fourier Transform. PACK$ and UNPAK$ convert between array data and the packed data format used by FFT$ and IFT$.

FFT$ and IFT$ (Fourier and inverse Fourier transforms)

```
CALL "FFT$",A$,B$,N
CALL "IFT$",A$,B$,N

A$:IO REAL PART OF INPUT SIGNAL
B$:IO IMAGINARY PART OF INPUT SIGNAL
 N:I  NUMBER OF POINTS IN TRANSFORM
```

FFT$ computes the Fourier transform of the signal in A$ and B$ and places the result in A$ and B$. A$ and B$ are integer arrays packed in the format given in appendix A. Since the transform produces fixed-point results, the Fourier transform normalizes on the forward transform to produce numbers in the range of the fixed-point storage array. This is similar to the Fourier series but unlike most discrete Fourier transforms in use, which usually normalize on the inverse transform.

Since two bytes in each string are used for each point in the transform, each string must be dimensioned to at least 2*N. The current length of the strings need not be 2*N, however; if it is less than this figure, the string will be filled with zeros until it is 2*N bytes long. A null string is acceptable; it will be filled with zeros before the transform is computed. The easiest way to compute the transform of a real signal in A$ would be to use the following instrucions:

```
B$=""
CALL "FFT$",A$,B$,N
```

The real portion of the output of the Fourier Transform is contained in A$ and the imaginary portion in B$. The output points are in the order usual for complex discrete Fourier transforms, shown later in this section.

IFT$ computes the inverse Fourier transform of the signal in A$ and B$ and places the result in A$ and B$. No normalization is done in IFT$, because FFT$ normalizes. The output of IFT$ is in the same format as the input of FFT$ and vice versa.

Note that for FFT$ and IFT$ N must be an integral power of two between 4 and 8192.

PACK$ and UNPAK$

```
CALL "PACK$",A$,A

A$:O  STRING TARGET FOR PACKED DATA
 A:I  ARRAY CONTAINING DATA TO BE PACKED

CALL "UNPAK$",A$,A

A$:I  STRING CONTAINING PACKED DATA
 A:O  TARGET ARRAY FOR UNPACKED DATA
```

PACK$ converts the floating point numeric data contained in the array A to fixed point format and places the output in A$. A$ must be dimensioned at least twice as large as A since two bytes are generated in A$ for each element in A. If any array element is less than -32768, -32768 is used in its place; if any element is larger than 32767, 32767 is used in its place. A$ need not be defined before PACK$ is called, but it must be dimensioned.

UNPAK$ converts the fixed point data contained in A$ to floating point data and places the output in A. A must be dimensioned at least one-half the current length of A$, since two bytes in A$ are used to compute each element in A.

Both PACK$ and UNPAK$ can accept multiple sets of parameters and will pack or unpack all of them. The parameters should be arranged in pairs of string,array. For example, CALL "PACK$",A$,A,B$,B,C$,C,D$,D is a valid command and will cause A to be packed in A$, B in B$, etc.

The fixed point and floating point formats are described in the following pages.

Fixed Point Format Used by Signal Processsing Routines

The integer representation of each floating number is equal to that number plus 32768. Thus, the range of integers that can be represented in the routines in the ROM pack is -32768 to +32767. Each numeric value occupies two adjacent positions in the string, the highest order byte being the first position. Some typical values and their equivalent representations are shown below:

FLOATING POINT VALUE	INTEGER REPRESENTATION (HEXADECIMAL)
+32767	FF FF
1000	83 E8
100	80 64
0	80 00
-100	7F 9C
-1000	7C 18
-32768	00 00

Ordering of Frequency Data

The frequency-domain data output from FFT$ or input to INF$ is in the following order:

ARRAY POSITION	STRING POSITIONS	FREQUENCY
1	1,2	0 (constant or d.c. term)
2	3,4	fundamental
3	5,6	2*fundamental
4	7,8	3*fundamental
.	.	.
.	.	.
.	.	.
N/2	N-1,N	(N/2-1)*fundamental
.	.	.
.	.	.
.	.	.
N/2+1	N+1,N+2	-(N/2)*fundamental
N/2+2	N+3,N+4	-(N/2-1)*fundamental
N/2+3	N+5,N+6	-(N/2-2)*fundamental
.	.	.
.	.	.
.	.	.
N-2	2N-5,2N-4	-3*fundamental
N-1	2N-3,2N-2	-2*fundamental
N	2N-1,2N	-fundamental

The fundamental is the frequency represented by N samples; thus fundamental=1/((time between samples)*N).

As in all complex Fourier transforms, if the input is a real signal, its power at each frequency except 0 is distributed evenly between the positive and negative components of the frequency, and thus is equal to the sum of the magnitudes of the power at the positive and negative components of the frequency. Note that the term corresponding to -(N/2)*fundamental has no positive counterpart, but, for real input functions, its counterpart is known to be its complex conjugate, as it true with all of the frequency components except zero.

Summary of Routines

```
 1. CALL "RTP"
 2. CALL "PTR"
 3. CALL "MULT"
 4. CALL "DIV"
 5. CALL "CPWR"
 6. CALL "CLOG"
 7. CALL "SINH"
 8. CALL "COSH"
 9. CALL "TANH"
10. CALL "ASINH"
11. CALL "ACOSH"
12. CALL "ATANH"
13. CALL "INTERP",X,Y,B,C[,C1]
14. CALL "INTEG",X,Y,B
15. CALL "DERIV",A,Z,A1,N2
16. CALL "PACK",A$,A,N,B
17. CALL "UNPACK",A$,A,N,B
18. CALL "ARYSET"
19. CALL "PLOT$",A$,D,I,S,N,B
20. CALL "FFT$",A$,B$,N
21. CALL "IFT$",A$,B$,N
22. CALL "PACK$",A$,A
23. CALL "UNPAK$",A$,A
```