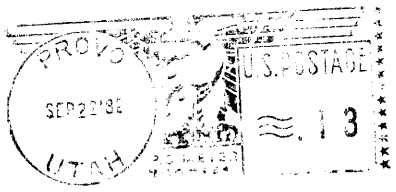
BINARY UTILITIES

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-BNU is a Binary Utility ROM Pack for the Tektronix 4050 Series Graphics Systems. It contains 20 machine laguage routines that are accessed through the BASIC CALL statement and are written in assembly language for the greatest possible speed of execution.

It has six I/O and graphics routines which can input from the GPIB into a string variable, plot binary string data, unleave binary string data, find the minimum and maximum values within a string of binary data, and reverse the order of byte pairs in a string of binary data. It has four data conversion routines which can convert between floating point data in an array and binary string data in a string variable and numeric data and ASCII 1's and 0's. It has 10 arithmetic and logic routines which can perform various arithmetic functions, logical operations, and shift operations on binary string data.

The routines in this ROM Pack are explained in the following manner. The format of the CALL statement is given, and each of the calling parameters 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. As far as possible, error checking is done by the routine to insure correct typing of the calling parameters.

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 a ROM Expander Unit. Press down gently until the edge card connector is seated in the receptacle connector.

I/O AND GRAPHICS ROUTINES

Introduction

There are six I/O and graphics routines in the 650/750-BNU. They are GPIBIN, PLOT$, UNLEAV, MAXI, MINI, and BSWAP. GPIBIN allows input from the GPIB. Data taken by this routine is stored in a string variable. PLOT$ draws a graph of data in a string variable. UNLEAV separates multiplexed data from a string variable. MAXI and MINI find the value and position of the largest and smallest datum in a string.

GPIBIN

CALL "GPIBIN",A$,N

A$:O Target for data
 N:I Number of bytes to take

The GPIBIN routine takes data from the GPIB and puts it in a string variable. This allows two computers to communicate with each other over the GPIB. The data should be sent with the WBYTES statement or the PRINT statement. All items received are put into the string, including addresses.

A$ is a string variable. It must be dimensioned large enough to contain the data to be read. If not previously dimensioned, it will be dimensioned to the default length of 72. N is the number of bytes to take. It may be a simple variable, an expression, or a literal value.

Example, sender:

```
100 PRINT @15:"THIS IS A TEST."
110 END
```

Example, reciever:

```
100 CALL "GPIBIN",A$,50
110 A$=SEG(A$,3,LEN(A$))
120 PRINT A$
130 END
```

Output:

```
THIS IS A TEST.
```

In this example, the computers are tied together with the GPIB cable. The sender outputs the primary listen address for device 15, which is 47, the secondary address for print, which is 108, and the string "THIS IS A TEST." The reciever puts all this data into A$. Line 110 strips off the addresses, and line 120 prints the received string.

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.

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.

UNLEAV

CALL "UNLEAV",A$,B$,I,S,N,B

A$:I Source string
B$:O Target string
 I:I Interval
 S:I Starting Point
 N:I Number of points to extract
 B:I Number of bytes per point

The UNLEAV routine extracts every I'th point in A$ starting at point S. It puts the extracted points in B$. It extracts until N points have been extracted. B is the number of bytes per point. B should be either 1 or 2. This is useful in separating multiplexed data.

Example:
```
100 A$="1234567890"
110 CALL "UNLEAV",A$,B$,3,2,2,1
120 PRINT B$
130 END
```

Output:
```
25
```

In this example, A$ contains ten characters. UNLEAV extracts two 1-byte points into B$ starting with the second point in A$ and using an interval of three. This extracts the digits 2 and 5. A$ is not changed.

```
CALL "MAXI",A$,M,P,B
CALL "MINI",A$,M,P,B

A$:I  Data string
 M:O  Target for value
 P:O  Target for position
 B:I  Number of bytes per sample
```

MAXI finds the value and position of the largest point in A$. MINI finds the value and position of the smallest point in A$. The number of bytes per sample (1 or 2) is specified by B. B may be a simple variable, an expression, or a literal value. The value of the extreme point is returned in M. The position in A$ is returned in P. M and P must be simple variables.

Example:

```
100 A$="123ABC0a123ABC"
110 CALL "MAXI",A$,M,P,1
120 PRINT "MAX",M,P
130 CALL "MINI",A$,M,P,1
140 PRINT "MIN",M,P
150 END
```

Output:

```
MAX     97      8
MIN     48      7
```

In this example, MAXI and MINI work on 1 byte data. MAXI returns the ASCII value and position of the character "a" and MINI returns the ASCII value and position of the character "0".

BSWAP

CALL "BSWAP",A$

A$:IO Data string

The BSWAP routine exchanges bytes in a string. It exchanges the first and second byte, the third and fourth bytes, and so on. The main purpose of this is to allow the other routines to manipulate data in which the least significant byte is first in the data string. If the length of the string is odd, the last byte is unaffected.

Example:

```
100 A$="1234ABCDE"
110 CALL "BSWAP",A$
120 PRINT A$
130 END
```

Output:

```
2143BADCE
```

In this example, BSWAP exchanges every 2-byte pair in A$. The length is odd, so the last byte is unaffected.

DATA CONVERSION ROUTINES

Introduction

There are 4 data conversion routines in the 650/750-BNU. They are PACK, UNPACK, DECBIN, and BINDEC. PACK and UNPACK convert between floating point data in an array variable and binary string data in a string variable. DECBIN and BINDEC convert between numeric data and ASCII 1's and 0's.

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

A$:IO Target or source string
 A:IO Target or source array
 N:I  Number of samples to convert
 B:I  Bytes per sample
```

The PACK routine compresses data from a floating point numeric array A to 1 or 2 byte integers and stores the binary result into string variable A$. The UNPACK routine takes data thus represented in string variable A$ and converts it to the 8 byte floating point format and stores each number in an element of the numeric array A.

N is the number of samples to convert. B is the number of bytes per sample. B should be either 1 or 2. Array A should be dimensioned to at least N elements. String A$ should be dimensioned to at least N*B bytes and the length should be at least N*B for UNPACK. If B is 1, the values PACKed can be no larger than 255. If B is 2, the values PACKed can be no larger than 65535.

Example:

```
100 DIM A(5),A$(10),B(10)
110 FOR I=1 TO 5
120 A(I)=I
130 NEXT I
140 CALL "PACK",A$,A,5,2
150 CALL "UNPACK",A$,B,10,1
160 PRINT B
170 END
```

Output:

```
 0      1      0      2
 0      3      0      4
 0      5
```

In this example, PACK packs five elements from array A into A$ using two bytes for each sample. UNPACK unpacks ten 1-byte samples from A$ into array B. This also demonstrates that the most significant byte is first.

BINDEC

CALL "BINDEC",E$,X

E$:I String to convert
 X:O Target variable

The BINDEC routine performs the opposite function of the DECBIN routine. The input string E$ can be any length up to 16 characters. The binary number represented by this string of ASCII 1's and 0's is converted into a decimal number and stored in variable X.

Example:
```
100 CALL "BINDEC","100010",X
110 PRINT X
120 END
```

Output:
```
 34
```

In this example, BINDEC converts the string "100010" to decimal and assigns the result, 34, to X.

ARITHMETIC AND LOGIC ROUTINES

Introduction

There are 10 arithmetic and logic routines in the 650/750-BNU. They are ADDB, SUBB, MULB, ANDB, ORB, EORB, ROLB, RORB, ASLB, and LSRB. ADDB, SUBB, and MULB perform arithmetic operations on binary string data. ANDB, ORB, and EORB perform logical operations on binary string data. ROLB, RORB, ASLB, and LSRB perform shift and rotate operations on binary string data.

ADDB

```
CALL "ADDB",B$,I,B
CALL "ADDB",B$,C$,B

B$:IO Target and string source 1
 I:I  Decimal source 2
C$:I  String source 2
 B:I  Bytes per sample
```

The ADDB routine allows adding a single integer value to an array of packed 1 or 2 byte binary numbers. If also allows two equal lengthed strings to be summed together. In the first case, I is the integer number which will be added to each word in B$. In the second case, C$ is a string of the same length as B$. Corresponding elements of C$ and B$ are added together. In both cases, the result is stored back in B$ and B is the number of bytes per sample and should be 1 or 2.

Example:

```
100 A$="123ABC"
110 CALL "ADDB",A$,2,1
120 PRINT A$
130 END
```

Output: 345CDE

In this example, ADDB adds 2 to each byte of A$. This increases each numeric character by 2 and changes each alphabetic character to the letter 2 later in the alphabet.

SUBB

```
CALL "SUBB",B$,I,B
CALL "SUBB",B$,C$,B

B$:IO Target and string source 1
 I:I  Decimal source 2
C$:I  String source 2
 B:I  Bytes per sample
```

The SUBB routine is used in the same way as the ADDB routine. The SUBB routine subtracts I or C$ from B$ and stores the difference back into B$.

Example:

```
100 A$="567XYZ"
110 CALL "SUBB",A$,2,1
120 PRINT A$
130 END
```

Output:

```
345VWX
```

In this example, SUBB subtracts 2 from each byte in A$. This decreases each numeric character by 2 and changes each alphabetic character to the letter 2 earlier in the alphabet.

MULB

```
CALL "MULB",B$,I,B
CALL "MULB",B$,C$,B

B$:IO Target and string source 1
 I:I  Decimal source 2
C$:I  String source 2
 B:I  Bytes per sample
```

The MULB routine is used to multiply a series of binary numbers in a string by an integer or by corresponding binary numbers in another string. As with ADDB and SUBB, the each element in B$ is multiplied by I or by the corresponding element in C$ and put back into B$. B is the number of bytes per sample and should be 1 or 2.

Example:

```
100 A$="12345"
110 CALL "MULB",A$,2,1
120 PRINT A$
130 END
```

Output: bdfhj

In this example, MULB multiplies each byte in A$ by 2. This doubles the ASCII value of each character.

ANDB

CALL "ANDB",B$,C$

B$:IO Target and source string 1
C$:I Source string 2

The ANDB routine allows two string variables to be ANDed together. Each byte of B$ is ANDed with the corresponding byte of C$, and the result is put back into B$.

Example:
```
100 A$="abcdqrst"
110 CALL "ANDB",A$,"^^^^>>>>"
120 PRINT A$
130 END
```

Output:
```
@BBD0224
```

In this example, ANDB performs a logical AND operation on the strings "abcdqrst" and "^^^^>>>>". This zeroes bits 6 and 1 of the first four characters of A$ and bits 7 and 1 of the last four.

CRB

```
CALL "ORB",B$,C$
```

B$:IO Target and source string 1
C$:I Source string 2

The ORB routine is used in the same way as the ANDB routine, but is performs a logical OR between the bytes in B$ and C$.

Example:

```
100 A$="1234ABCD"
110 CALL "ORB",A$,"@@@@    "
120 PRINT A$
130 END
```

Output:

```
qrstabcd
```

In this example, ORB performs a logical OR operation on the strings "1234ABCD" and "@@@@   ". This sets bit 7 of the first four characters in A$ and bit 6 in the last four.

EORB

CALL "EORB",B$,C$

B$:IO Target and source string 1
C$:I Source string 2

The EORB routine is also used in the same way as the ANDB and the ORB routines with the exception that is performs a logical EXCLUSIVE OR function.

Example:

```
100 A$="QRST5678"
110 CALL "EORB",A$,"^^^^^^^^"
120 PRINT A$
130 END
```

Output: 1234UVWX

In this example, EORB performs an EXCLUSIVE OR function on the strings "QRST5678" and "^^^^^^^^". This inverts bits 6 and 7 of each byte of A$ exchanging numbers and upper case letters.

ROLB

CALL "ROLB",B$,C,N,B

B$:IO Data string to be rotated.
 C:IO Initial carry value and target for final carry
 N:I Number of bit places to shift
 B:I Number of bytes per sample

The ROLB routine is used to perform a Rotate Left bit shift over 1 or 2 byte binary words. The rotate function causes the carry generated from the previous shift to be shifted into the least significant bit position. The carry may be initially set or cleared by argument C which should be 1 or 0. The value of C will be used for the carry on the first shift in each word but will be returned with the actual value of the carry resulting from the final shift on the last binary word. N is the number of bit places to shift. B is the number of bytes per binary word over which shifting is to take place and should be 1 or 2.

Example:

```
100 A$="123"
110 C=1
120 CALL "ROLB",A$,C,1,1
130 PRINT A$,C
140 END
```

Output:

```
ceg     0
```

In this example, ROLB performs a left rotate operation on the string "123" with an initial carry of 1. This essentially multiplies the ASCII value of each character by 2 and adds 1.

RORB

CALL "RORB",B$,C,N,B

B$:IO Data string to be rotated.
 C:IO Initial carry value and target for final carry
 N:I Number of bit places to shift
 B:I Number of bytes per sample

The RORB routine is the same as the ROLB routine except that the direction of the shifting is to the right.

Example:
```
100 A$="ceg"
110 C=0
120 CALL "RORB",A$,C,1,1
130 PRINT A$,C
140 END
```

Output:
```
123     1
```

In this example, RORB performs a right rotate operation on the string "ceg" with an initial carry of 0. This is the inverse operation of the example for ROLB. It essentially divides the ASCII value of each character by 2.

ASLB

CALL "ASLB",B$,C,N,B

B$:IO Data string to be shifted
C:O Target for final carry
N:I Number of bit places to shift
B:I Number of bytes per sample

The ASLB routine is an Arithmetic Shift Left operation the same as the ROLB routine except that on each shift a 0 is shifted into the least significant bit position instead of the result from the carry. The variable C is returned with the value of the carry from the last shifting operation. N is the number of bit places to shift. B is the number of bytes per word over which the shifting is to take place and should be 1 or 2.

Example:

```
100 A$="123"
110 CALL "ASLB",A$,C,1,1
120 PRINT A$,C
130 END
```

Output:

```
bdf     0
```

In this example, ASLB performs a left shift operation on the string "123". This essentially multiplies the ASCII value of each character by 2.

LSRB

CALL "LSRB",B$,C,N,B

B$:IO Data string to be shifted
C:O Target for final carry
N:I Number of bit places to shift
B:I Number of bytes per sample

The LSRB routine is a Logical Shift Right operation which is the same as the RORB routine except that it shifts a 0 into the most significant bit position rather than the result of the carry. It is also the same as the ASLB routine except that the direction of the shifting is reversed.

Example:

```
100 A$="bdf"
110 CALL "LSRB",A$,C,1,1
120 PRINT A$,C
130 END
```

Ouput:

```
123     0
```

In this example, LSRB performs a right shift operation on the string "bdf". This essentially divides the ASCII value of each character by 2. Note that this is the inverse of the example for ASLB.

SUMMARY OF ROUTINES

```
 1. CALL "GPIBIN",A$,N
 2. CALL "PLOT$",A$,D,I,S,N,B
 3. CALL "UNLEAV",A$,B$,I,S,N,B
 4. CALL "MAXI",A$,M,P,B
 5. CALL "MINI",A$,M,P,B
 6. CALL "PACK",A$,A,N,B
 7. CALL "UNPACK",A$,A,N,B
 8. CALL "DECBIN",A$,B$,X
 9. CALL "BINDEC",A$,X
10. CALL "ADDB",A$,I,B
    CALL "ADDB",A$,B$,B
11. CALL "SUBB",A$,I,B
    CALL "SUBB",A$,B$,B
12. CALL "MULB",A$,I,B
    CALL "MULB",A$,B$,B
13. CALL "ANDB",A$,B$
14. CALL "ORB",A$,B$
15. CALL "EORB",A$,B$
16. CALL "ROLB",A$,C,N,B
17. CALL "RORB",A$,C,N,B
18. CALL "ASLB",A$,C,N,B
19. CALL "LSRB",A$,C,N,B
20. CALL "BSWAP",A$
```