FBX BASIC

Reference Manual

PROGRAMMA

Software Program Products

2000.001

October 1977 Edition

This edition (2000.001) is a major revision and obsoletes all previous editions and documents.

Technical changes are marked with a bar in the outer margin. Changes due to subsequent releases will be documented in future publications bulletins or revisions.

Requests for copies of PROGRAMMA publications should be made to your PROGRAMMA representative or to the PROGRAMMA central office.

A reader's comment form is provided at the back of this publication. If the form has been removed, comments may be addressed to PROGRAMMA CONSULTANTS, Publications Dept., P.O. Box 70127, Los Angeles, CA 90070.

PREFACE

This reference publication is intended for programmers using the PROGRAMMA CONSULTANTS FBX BASIC Language. This publication describes how to write BASIC source statements, system start-up, alteration of system parameters, and error handling.

The reader should be familiar with the hardware manuals and operational procedures for the devices in his particular computer configuration.

TABLE OF CONTENTS

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

I. INTRODUCTION

To use a computer, the user must learn a language the computer understands. Sphere M6800 computer systems understand several languages. Most of these are meant for some special purpose, such as the solution of scientific, engineering, or business problems. BASIC is intended as an all-purpose language, hence the name; Beginners All-purpose Symbolic Interpretive Code.

Because of its similarity to English, BASIC is a good language for users who are not interested in the internal workings of the computer.

This manual has been written as a tutorial/reference guide to Programma Consultants Full BASIC Extended. The chapters are logically ordered and concise on every topic necessary to the beginner and experienced user alike.

The sections on BASIC instructions and commands are easily referenced and clearly indexed to make the information quickly available. All functions, commands and statements are also listed in the appendix "BASIC Quick Reference".

This manual was designed and written for FBX 2.0, as new versions of FBX are issued updated pages of this book will also be available. It is recommended that the user keep these pages in a three-hole punch notebook. Updates can then be easily added and obsolete pages removed.

Developed for Programma
Consultants by

M. Higgins

&

A. Rosenthal

II. IMPLEMENTATION OF BASIC

System requirements for execution of Full BASIC Extended:

MEMORY	At least 12K RAM.
INPUT DEVICE	Keyboard
OUTPUT DEVICE	CRT
STORAGE DEVICE	Cassette I or II.
OPTIONAL DEVICES	Printer, Disk reader, Plotter, Paper tape reader punch.
SOFTWARE	FBX and Sphere PDS[1] on PROM.
OPTIONAL SOFTWARE	OS/1[2] for Disk reader.

Implementation of BASIC from floppy disk is achieved in the following manner:

1. System powered up and Reset.
2. Disk reader power on and diskette in reader.
3. Load boot strap program IP from Cassette (see appendix "Cassette Loading Procedure").
4. Execute program IP at location 0400H.
5. CRT will display "$" prompt character.
6. Type "BASIC" and press the ESCAPE key.

When the cursor returns BASIC is loaded and running. It is now ready to accept commands or instructions. These include all the disk only commands; DUMP, RESTORE, and Cntl E.

[1] PDS - Sphere Program Development System Monitor on EPROM

[2] OS/1 - Disc Operating System Monitor for the ICOM FD-360 Floppy

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

IMPORTANT INFORMATION.

PRINT DRIVER ROUTINE	TTY	OKI	PR-40
File Name	BASIC	BASIC	BASIC
File Type	OBJ	OBJ	OBJ
Load Address	0600	0600	0600
End of Code	1EE7	1FD2	1EF0
Execution Address	0600	0600	0600
Soft Start[1] Address	0602	0602	0602
Memory Limit Pointer[2]	1B0F	1B0F	1B0F

[1] Should your BASIC program abend abnormally, or should you be forced to reset the computer Central Processing Unit (CPU), open this location from PDS DEBUG and type Cntl G. The current BASIC source program in the memory will be intact.

[2] This location and the one following it will contain the physical end of RAM memory that is found in your machine. This pointer may be changed to any reasonable value, if you desire to limit the memory requirements of the BASIC Interpreter. Under normal operating conditions, the end of memory limit pointer should not be set to any value less than 8K (H 1FFF).

For implementation of BASIC from Cassette I or II, see appendix "Cassette Loading Procedure" with the following information:

PRINT DRIVER ROUTINE	TTY	OKI	PR-40
File Name	BT	BO	BS
File Type	CASS	CASS	CASS
Load Address	0600	0600	0600
End of Code	1EE7	1FD2	1EF0
Execution Address	0600	0600	0600
Soft Start Address	0602	0602	0602
Memory Limit Pointer	1B0F	1B0F	1B0F

III. DATA REPRESENTATION

A. CONSTANTS

1. NUMERIC

A numeric constant consists of a string of digits that represent an unchanging amount or number. For example; the number of eggs in a dozen is said to be constant at 12. The force of gravity on the surface of the earth is constant at 9.8 meters per second squared. The number of hours in a day is constant at 24.

Numeric constants may also have a decimal portion. Such as the representation of the sales tax rate in the state of California at 0.06. Another example is the base of natural logarithms at 2.714818.

All numeric constants have another attribute, that of sign; a plus sign (+) for values greater than zero and a minus sign (-) for values less than zero. The absence of any sign indicates a value greater than zero.

In this version of BASIC numeric constants have a limited range. Numeric constants must fall within the limits of 99999999 to -99999999 inclusive.

FBX has one predefined numeric constant. The letters PI are the same as using the numeric constant 3.1415926.

Examples of Valid Numeric Constants

6	123.75	0	.0038	PI	-67	18
3.14159	1.4143		-.000003	0.00	0675	

Examples of Invalid Numeric Constants

46,312	no commas allowed
$23.60	no dollar sign allowed
90456739854	exceeds range
-+2345	only one sign per number allowed
.0154	no leading zero preceeding decimal point

All numeric constants that may contain a decimal point are subjected to the following syntax restriction:

If the numeric constant represents the fractional part of a whole number, then the decimal point of the numeric constant must be preceeded by the numeric zero.

For Example:

.0154 would be represented as 0.0154

A. CONSTANTS

2. LITERAL

A literal (string) constant is a set of Characters* enclosed in double quotation marks ("). In BASIC a string constant may consist of from 0 to 255 characters. A name is a good example of a string constant:"FARRAH FAWCETT-MAJORS". The only restriction on the contents of a string constant is that it may not contain a double quote mark.

Examples of Valid Literal Constants

"A.H. ROSENTHAL" "1600 PENN. AVE." "* (##) *IS HERE "

Examples of Invalid Literal Constants

"MELVIN" NORRELL"	string contains quote mark
" " "	string also contains quote
NOW IS THE TIME FOR	missing quote marks

*ASCII Character set.

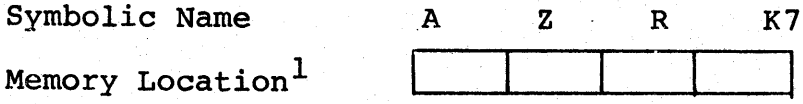
American Standard Code for Information Interchange

B. VARIABLES

1. NUMERIC

Numeric variables are used to store changing values. The time of day, the score in a game, and the stock market prices are all variable quantities.

Variables are locations in the computer's memory. These locations are referenced by symbolic names. The diagram below illustrates a simple symbol table of memory locations with symbolic names.



Numeric variables are used by the programmer to store numeric constants, but unlike constants, numeric variables can be altered by the program. This is accomplished by the use of the correct symbolic name.

Symbolic names for numeric variables may consist of just a single letter of the alphabet (A,B,C,...Y,Z), or a single letter followed by a single digit (0 to 9). Using all the possible combinations, there are 286 legal symbolic names for numeric variables.

Numeric variables have the same value range limits as those found for numeric constants. Numeric variables may have a sign specified in the same manner as a numeric constant.

Examples of Valid Numeric Variables

X A1 Z9 T Q0 C7

[1] Memory locations are taken from program storage given to variable storage as each symbolic name is encountered.

Examples of Invalid Numeric Variables

1	must begin with letter
XX	cannot contain two letters
A2B	only single digit after letter
*3	must begin with letter (A-Z)
APLHA	only single letter allowed

B. VARIABLES

2. ARRAYS

A numeric variable with more than one storage location associated with its name is called an array. Each separate value in an array is called an element of that array. The diagram below illustrates an array called A, which contains five elements.

Memory locations

The Array

A

1	2	3	4	5

The array above is a one-dimensional array called a vector. A single column of elements with a common symbolic name is termed a vector. A two-dimensional array is called a matrix. A matrix is shown below.

Memory locations

The Matrix

Z

	Col 1	Col 2	Col 3	Col 4
Row 1	1,1	1,2	1,3	1,4
Row 2	2,1	2,2	2,3	2,4
Row 3	3,1	3,2	3,3	3,4

The array Z is a 3 by 4 matrix, containing 12 separate elements. When referring to any two-dimensional array it is described in terms of rows by columns.

Every element of an array can be independently accessed by subscripting the array name. For example: Z(1,3) refers to the element at row one, column three. In the array A shown before, A (3) would name the third element from the top, where A(1) would be the first or top. It should be noted that A(6) would be an error, as the array A has only five elements.

Each element of an array is a numeric variable. All the attributes and rules which apply to a numeric variable, apply to the elements of an array.

The rules for array symbolic names are different from those which apply to simple numeric variables. The symbolic name of an array must be a single letter of the alphabet, no digit is allowed. A and A (3) are two different variables.

The subscripts of an array must be positive integers greater than zero, and must define the array. For example: take the array Z from before, the symbolic name Z must have two subscripts. Like Z(1,2) or Z(3,3)

Examples of Valid Arrays

A(10,12)	defines a matrix of 120 elements in 10 rows of 12 columns
K(5)	specifies a 5 element vector
R(1,120)	this also defines a 120 element matrix

Examples of Invalid Arrays

D(3,4,5)	too many subscripts, 2 max.
R3(5)	illegal symbolic name, not allowed
Z(3.5)	non-integer subscript not allowed
P(0,3)	both subscripts must be greater than zero

B. VARIABLES

3. LITERAL

A literal variable is a series of contiguous memory locations, referenced by a symbolic name, and used to store any combination of alpha-numeric characters. The contents of a literal variable can change during program execution. Literal variables can be used to store literal contants in much the same manner that numeric variables can be used to store numeric contants.

Symbolic names for literal variables are subject to the following restrictions: they must be a single letter of the alphabet followed by a dollar sign ($) character.

The amount of storage assigned to a literal variable is determined by the programmer. A literal variable can never contain less than one or more than 255 characters.

Examples of Valid Literal Variables

A$ C$ H$ Z$ K$ S$

Examples of Invalid Literal Variables

A0$	must be a single letter, no digit allowed
B4	single letter must be followed by dollar sign
5$	first character must be a letter (A to Z)

IV. ARITHMETIC EXPRESSIONS

A. NUMERIC EXPRESSIONS

1. OPERATIONS

A numeric expression is a series of constants and/or variables combined with arithmetic operations to form a logical statement. For example: 5-3, read five minus three, is a correct expression.

In BASIC the following operators may be used in an arithmetic expression:

Operator	Operation	Example of use
+	Unary Positive	+5
-	Unary Negative	-D
*	Multiplication	6*8
/	Division	234/45
+	Addition	36+82
-	Subtraction	639-987

The first two operators in the table are used on a single variable or constant to specify the sign, hence the term unary (meaning single or one). In unary expressions the plus sign (+) is optional. The last four operators in the table are binary operators requiring two values to work on.

In arithmetic expressions there can only be one operator between two values and only one value between two operators.

Examples of Valid Use of Operators

Operator	Explanation
-3	negative three
+5	positive five
3*A	three times the value in A
6/2	six divided by two

Examples of Invalid use of Operators

Operator	Explanation
--6	only one operator may precede a number
4R	missing operator; not the same as 4*R
6*-B	only one operator between two values
6 X 7	illegal operator

A. NUMERIC EXPRESSIONS

2. ORDER OF OPERATION

Evaluation of large arithmetic expression, i.e. those with two or more operators, follows a logical order of operation.

In the table below the operators are listed showing the order of precedence.

Operator	Meaning	Order of Operation	
+	Unary Plus	1	(Highest)
-	Unary Minus	1	
*	Multiplication	2	
/	Division	2	
+	Addition	3	
-	Subtraction	3	(Lowest)

As the table indicates, during the evaluation of an arithmetic expression all Unary operators and their operands are computed first. Next all Multiplication and Division and finally all Addition and Subtraction operations.

The BASIC evaluation of an arithmetic expression is left to right by order of operation. When two or more operations of the same precedence appear in a single expression they are computed at the same time.

Examples of BASIC Evaluation of Arithmetic Expressions

Expression	Evaluated as	Explanation
1+2*3	7	multiplication first addition second
12/2*4	24	left to right
5*2-9/3	7	multiplication first division second subtraction third

To alter the sequence of normal evaluation in an arithmetic expression the parenthesises () are added to the top of the list of legal operators. When used, the parenthesis have the highest value in order of precedence. Parenthesis must occur in left right pairs, for every open there must be a close.

Portions of arithmetic expressions isolated by a set of parenthesis are operated on first. All sets of parenthesis must have a sign or operator. For example: (3/4) * (6-7) or -(3)-(3/4).

In the following expressions, the operators and operands remain the same and parenthesis are used to alter the order of evaluation:

Expression	Evaluation
1+2*3/4	2.666
(1+2)*3/4	2.25
(1+2*3)/4	1.75

A. NUMERIC EXPRESSIONS

3. EQUATIONS

In BASIC an equation is defined as a numeric variable, followed by an equal sign, followed by an arithmetic expression. The right side of the equal sign, the expression is evaluated and the results stored in the variable on the left of the equal sign. There can never be anything but a numeric variable on the left of the equal sign. Unlike algebraic equations, all the variables on the right side of the sign must be defined and have a value.

Examples of Legal Equations

Equation	Evaluation of Equation
A=2+6+8/2+6	A=18
B1=12/2*3	B1=18
R(3)=12/(2*3)	R(3)=2
J(1,6) = -5+2	J(1,6)=-3
A=(2+4)/(-5)+23)	A=.33333
Q=Q+1	Value in Q equals value plus one

Examples of Illegal Equations

Equation	Error
R=3-/6	too many binary operators
K=(6*3) (2-5)	missing parenthesis operator
L6=(3-(2*6)	missing right parenthesis
F(2)=3/(4-1)	division by zero
3/6=A	variable on wrong side
P(3,78)=AJ/16	AJ not legal variable

B. LITERAL EXPRESSIONS

1. EQUATIONS

A string (literal) equation consists of a literal variable, followed by an equal sign, followed by either a string constant or another string variable. Example: A$ = "THE IBM CORP." Before a string variable can be used in a program, the programmer must determine the length, or the number of characters that will be used with the string variable. A special BASIC statement called the DIMension statement is used to let BASIC know the size of a string variable.

Literal equations are used to assign values, either constant values or the values of other variable strings, to legal variables. Note that a literal equation is commonly refered to as an assignment or LET statement.

There are no string equation operators available in BASIC. The only valid statement to the right of the equals sign is either a constant or a single variable.

Examples of Legal Literal Equations

```
A$ = "THIS IS A BASIC STRING"

C$ = A$

Z$ = "DARTH VADAR LIVES!"
```

Examples of Illegal Literal Equations

A$ = B$ - 3	no operators allowed
C$ = "16" + "2"	no operators or operations allowed
A = C$	the string variable to the left of the equals sign cannot be a numeric variable

C. INTRINSIC FUNCTIONS

Many arithmetic operations are needed so commonly in programming that BASIC has some of these operations as predefined functions. When the user wishes to use one of these functions he types the two, three or four character function name followed by a numeric argument if required. Functions may be placed anywhere that a numeric constant or variable may be used.

The following numeric functions are supported by BASIC:

PEEK

The PEEK function allows the user to examine a single memory location in the computer. The PEEK function takes a positive integer in base ten and returns the value that is stored at that location in base ten. The form of the PEEK function is...

PEEK (address)

ABS

The ABS function computes the absolute value of a numeric argument. The ABS function operates on any numeric constant, variable or expression. The form of the ABS function is...

ABS (numeric value)

SGN

The SGN function returns the sign of the numeric argument. If the argument is positive a positive one (1) will be returned, if the argument is negative a negative one will be returned and if the argument is zero a zero will be returned. The form of the SGN function is...

SGN (numeric value)

INT

The INT function will take a numeric argument and truncate the decimal portion and return only the integer portion. The INT function has the following form...

INT (numeric value)

RND

The RND function will return a random number greater than zero and less than one. The RND function requires a numeric argument, however, the argument is a dummy value and is not used in the computation. The form of the RND function is...

RND(0)

PI

The PI function does not use a numeric argument. Its purpose is to return the value and pi 3.1415926 to the number of digits desired. The form of the PI function is...

PI

The following string functions are supported by BASIC:

LEN

The LEN function will return the length of the string used in the argument. The target variable for this function must be a numeric variable or array element. The form of the LEN function is ...

LEN (literal variable)

STR

The string function has the form...

STR (string, start, length)

In the string function the "string" is a literal variable it can be the same variable as the target (left of the equal sign); "start" is a numeric constant which describes where in the string to start (which character); "length" is again a numeric constant which tells how many after "start" are to be moved or compared.

STR is the only BASIC function which may occur on either or both sides of the equal sign.

Examples of the use of STR

Example	Results of function
A$=STR(B$,3,5)	characters 3 to 7 of B$ are placed in A$

V. BASIC PROGRAM STRUCTURE

A. SIMPLE PROGRAMS

A BASIC program is a series of predefined statements or instructions that are executed in a specific order.

When using BASIC the programmer gives each instruction a line number. BASIC instructions are executed in ascending numerical order, except thru modification of the program structure by specific instructions for this purpose.

Execution of a BASIC program is similar to reading a book. One starts at page one and continues in order unless instructed to reference another part of the book.

When entering BASIC programs into the computer the programmer is not required to enter the lines in numerical order. Instead the computer will arrange the lines in numeric order just prior to execution. For example, the programmer might enter line number 10 and then line number 20. If he then wishes to place a line between 10 and 20 he could enter line number 15 (or any number between 10 and 20).

To change a line once it is stored the programmer need only type the line number followed by the new text. For example:

```
5 REM
20 PRINT
30 GOTO 5
40 END
20 PRINT "HELLO"
```

would list as:

```
5 REM
20 PRINT "HELLO"
30 GOTO 5
40 END
```

To remove a line once it is stored the programmer simply types the line number followed by a carriage return. For example:

```
5 REM
23 GOTO 50
30 PRINT
23
```

would list as:

```
5 REM
30 PRINT
```

In BASIC there are two types of instructions, statements and commands. Statements are proceeded by line numbers and are grouped together and stored as a BASIC program. These statements require no immediate action by the computer.

Commands are not proceeded by line numbers and are executed immediately by the computer. They are not stored.

Once a BASIC program has been entered into the computer and completed the programmer may issue the RUN command and the computer will execute the program.

B. MULTIPLE STATEMENTS PER LINE

The one exception to the standard rules presented so far is the multiple statements per line feature of BASIC. Two or more statements may be placed on the same line number if they are separated by a back slash (\ ) character. When that line is executed all the statements will be executed in left to right order. The general form of the multiple statement per line is...

line number statement \ statement \ statement...

where...

line number :
is any legal BASIC statement.

For example:

```
10  X=0\R=9\IF  P=10  GOTO  30

234  PRINT\PRINT\PRINT
```

are correct usage of multiple statements per line feature.

The following are examples of illegal usage of the multiple statement per line feature:

```
20   X=234 PRINT          <-----missing back slash
```

The END, GOTO, and RETURN statements must appear at the end of a multiple statement because none of the statements after them will ever be executed.

C. SYMBOLS USED

The following symbols are used throughout this manual to help explain BASIC statements and commands:

Symbol	Meaning
b	indicates a space character is used here
cr	indicates a carriage return is used here
'line number'	any legal line number is used here; line numbers must be integer values between 1 and 65535 inclusive
A^c or cntl A	super script c or cntl indicates a control character is used here
ESC	ESCape character
{ } braces	used to enclose required attributes
[ ] brackets	used to enclose optional attributes

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

VI. FULL BASIC EXTENDED: STATEMENTS

Version 2.0

CLEAR statement;

The CLEAR statement is very useful in formatting output. Its function is to clear the CRT screen of all printing and then to position the blinking cursor at the upper left-hand corner of the screen. The general form of the CLEAR statement is...

```
line number   CLEAR
```

where...

line number :
is any legal line number.

No additional arguments are needed for the CLEAR statement. An example of the use of the CLEAR statement is...

```
10  CLEAR
20  INPUT A
30  CLEAR
40  PRINT A,A*A, A*A*A
50  GOTO  10
60  END
```

This program will allow the user to enter a number, CLEAR the screen, print out the number and it's square and it's cube, CLEAR the screen again and repeat the cycle.

DIM statements:

The DIM statement is used to set aside computer memory for the storage of arrays or strings. Whenever an array or literal string is to be used in a program it must be DIMensioned. The DIM statement tells the computer the name, length and size of each subscript of the array or string. The DIM statement has the following general form...

line number DIM symbolic name (dimension)

where...

line number :
is any legal line number.

symbolic name :
is any single letter (A-Z) for arrays or any single letter (A-Z) followed by a dollar sign ($) for strings.

dimension :
is a single integer greater than zero and less than 256 for literal strings or a single integer greater than zero for single dimensioned arrays called vectors or two integers greater than zero for a two dimensional array called a matrix.

The largest string possible is 255 characters. The largest array possible is dependent on the amount of memory on the computer.

The following DIM statements...

```
10 DIM W(10) , T(5 , 5)
20 DIM R$(72)
```

will set aside memory for an array called W that is a vector ten elements long, an array called T with two dimensions five long each that will hold 25 elements and finally a literal string called R$ that will hold 72 alphanumeric characters.

DUMP statement:

The DUMP statement is fully explained under the commands section of this manual under DUMP/RELOAD. The user is instructed to check this section for a full explanation of the use of the DUMP statement.

The DUMP statement can be used under program control to save the symbol table and dynamically allocated variables to cassette or disk. The general form of the DUMP statement is as follows:

```
line number   DUMP  XX
                    disk file name
```

where...

XX is any valid cassette file name

disk file name is any valid disk file of type D for DATA

END statement:

The END statement causes termination of a BASIC program. Upon finding the END statement the computer will stop execution of the program currently running and wait for further instructions from the operator thru the keyboard.

The END statement is usually the highest numbered, and therefore last statement in a BASIC program. The END statement however can be placed anywhere in a BASIC program. The general form of an END statement is...

line number END cr

where...

line number :
is any legal line number.

Examples of Valid END statements are:

```
9999  END
65535 END
```

FOR . . . NEXT statements:

A group of statements in a program that are executed repeatedly are called loops. A GOTO statement may be used to make an infinite loop, that is a loop that repeats as long as the computer is on. Sometimes, however, theprogrammer only needs to execute a loop a certain number of times. This can be done with a FOR ...NEXT loop. The FOR statement has the following general form...

line number FOR index variable = lower limit TO upper limit

where...

line number :
is any legal line number.

index variable :
is any single letter or any single letter followed by a single digit.

lower limit :
any legal numeric constant, variable or expression.

upper limit :
any legal numeric constant, variable or expression that is greater than or equal to the lower limit.

The NEXT statement has the following general form...

line number NEXT index variable

where...

line number :
is any legal line number.

index variable :
is any single letter or any single letter followed by any single digit.

When the FOR statement is encountered the computer will set the index variable to the value of the lower limit. Control then goes to the statement following the FOR statement. Execution proceeds normally until the NEXT statement is reached. The NEXT statement causes the value in the index variable to be incremented by one and compared with the value in the upper limit. If the value in the upper limit is greater than the value in the index variable control is passed to the statement following the FOR statement and the loop continues. If the value in the upper limit is less than or equal to the value in the index variable control is passed to the statement following the NEXT statement and the loop is said to have terminated.

An example of a FOR. . . NEXT loop is. . .

```
10  FOR I=1 TO 10
20  PRINT I
30  NEXT I
40  END
```

This program will print the sequence 1,2,3,4,5,6,7,8,9,10.

FOR . . . NEXT loops may also be nested, that is, a loop maybe placed inside another loop. As example...

```
10  FOR I=1 TO 10  ----------
20  FOR J=2 TO 22  -------   |
30  FOR K =39 TO R*4 --  |   |
40  PRINT I,J,K  NEST I II III
50  NEXT K           --  |   |
60  NEXT J           -------  |
70  NEXT I      ------------  |
80  END
```

The programmer should be careful not to jump into the middle of a FOR . . . NEXT loop outside the loop. If a NEXT statement is encountered before a matching FOR statement is executed an error will result. Also the programmer should be careful not to accidently change the index variable inside the FOR . . . NEXT loop since this could cause unpredictable results.

GOSUB . . . RETURN statements:

It is often necessary for a group of statements to be executed at several points in a program. One way of doing this is with a FOR. . . NEXT loop and another way is with a GOTO statement. A third way to do this is by setting up a group of statements as a subroutine and call that subroutine from several different points in the program. A subroutine is called by executing a GOSUB statement. GOSUB statements work almost like a GOTO statement but in addition to a transfer of control to another part of the program the line number of the statement following the GOSUB is saved. The subroutine is then executed until a RETURN statement is encountered. The RETURN statement will cause the computer to jump to the line after the calling GOSUB statement. The general form of the GOSUB statement is...

line number GOSUB subroutine location

where...

line number :
is any legal line number.

subroutine location :
is any legal line number.

The general form of the RETURN statement is...

line number RETURN

where...

line number :
is any legal line number.

The return addresses (line number of statement following calling GOSUB statement) are saved on a last-in-first-out basis. This use of a last-in-first-out stack allows subroutines to call other subroutines. For example, the main program might call a subroutine that computes square roots. The square root subroutine might call another subroutine that will get input from the user. When the input routine is completed the RETURN statement will return control to the square root routine. When the square root routine is completed the RETURN statement will return control to the main program.

GOTO statements:

Very often, in programming, it is necessary to have the computer repeat a block of statements. For instance, a program to add pairs of numbers together might look like this...

```
10  INPUT A,B
20  PRINT A;"PLUS";B;"EQUALS";A+B
30  END
```

This program will work fine unless the user wishes to add more than one pair of numbers together. Each time another pair must be added the program must be RUN again.

BASIC allows the programmer to make unconditional transfers of control from one part of the program to another. This is done with the GOTO statement. The GOTO statement has the following general form...

```
 line number  GOTO    destination
```

where...

line number:
is any legal line number.

destination :
is any legal line number.

GOTO statements do exactly what the word implies. They GOTO another part of the program. The line number that control is to be passed to is the destination.

In the addition program above the following GOTO statement may be added...

```
25  GOTO 10
```

The program now reads...

```
10 INPUT A,B
20 PRINT A;"PLUS";B;"EQUALS";A+B
25 GOTO 10
30 END
```

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

This program will allow the user to INPUT two numbers, PRINT the two numbers and their sum and GOTO the INPUT statement again. In this program the END statement is never executed instead the program is in an infinite loop that can only be stopped by resetting the computer or by entering a CNTL A command during the INPUT statement.

Examples of Valid GOTO Statements:

```
100 GOTO 2505

534  GOTO 600

950  GOTO 65535.
```

IF statements:

IF statements allow the program to branch to one of two statements depending on the outcome of a predefined condition. IF statements are called conditional branching instructions as compared to GOTO statements which are called unconditional branching instructions. The general form of an IF statement is...

line number IF condition statement

where...

line number :
is any legal line number.

condition :
is any numeric expression followed by any relation operator followed by a numeric expression or any literal value followed by any relation operator followed by a literal value.

statement :
is any BASIC statement that may follow an IF statement.

The condition part of the IF statement is a test to compare one value with another. The values may be numeric or literal however a numeric value may not be compared with a literal value. The following are legal relation operators...

Operator	Definition
=	equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
<>	not equal to

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

The statement part of the IF statement is executed if and only if the condition part of the IF statement is true. If the condition part of the IF statement is not true control will pass to the next line of the program and the statement part will be skipped over. The following may be used as the statement part of an IF statement...

CLEAR	clears CRT
PRINT	print output
LET	assign value to variable
GOTO	transfer to another part of program
GOSUB	execute subroutine
RETURN	return from subroutine
END	stop execution of program

Examples of Valid IF statements:

```
100 IF A-56 = Y  GOTO 990
554 IF A$ = "YES" PRINT "DONE"  END
20 IF A >= 0.004  RETURN
7770 IF H$=I$  CLEAR
```

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

INPUT statement

Most BASIC programs are designed to operate on several sets of different data. It is therefore useful to have the program request keyboard data entry. This is done by using the INPUT statement. The general form of the input statement is...

line number INPUT variable list

where...

line number :
is any legal line number.

variable list :
is any group or combination of numeric or literal variables, separated by commas.

When the INPUT statement is executed by the computer a flashing cursor will appear on the CRT as a prompt for data input. At this time the user should enter the required data, if more than one piece of data is requested, commas should be used to separate the values. When all the requested data has been entered the user types a cr and program execution will continue. The variables listed in the variable list now have values as assigned by the INPUT statement.

Examples of Valid INPUT statements:

```
10  INPUT A

285 INPUT A$

1000 INPUT R$,X,Y,Z(1),Z(2),Z(3)
```

The following BASIC program demonstrates the use of the INPUT statement:

```
10  INPUT A
20  LET B=A*A
30  PRINT A;  "SQUARED IS ";B
40  END
```

The program allows the user to enter any legal BASIC numeric constant, and receive as output the square of that number.

LET statements:

It is often necessary to assign values to variables during program execution. The way that this is done in BASIC is to use the LET statement. The LET statement is used to assign the value of another variable to a specified variable, or to assign a constant value to a variable. It is assumed that the variables mentioned above are of the numeric type. Conversely, the LET statement is also used to assign the string value of another string variable to a string variable, or to assign a string constant to a string variable.

The general form of a numeric LET statement is...

line number LET numeric variable = numeric expression

where...

line number;

is any legal line number.

numeric variable;

is any numeric variable.

numeric expression;

is any numeric constant, variable, or expression.

BASIC will evaluate the expression on the right side of the equals sign and proceed to store the resulting value in the numeric variable on the left side of the equals sign. For instance, look at the following BASIC LET statement...

```
10  LET A=5*3-1
```

it will compute the value of the expression 5*3-1 for a result of 14, and it will store that result in the numeric variable called A.

BASIC allows the numeric variable that appears on the left of the equal sign to also appear on the right side of the equal sign. Look at the following statement...

```
250  LET X1=X1-1
```

it directs the computer to "take the value stored at location called X1, subtract one from it, and proceed to store the

result in the location called X1."

LET statements may also be used to assign literal values to literal variables. The general form of a string LET statement is as follows...

line number LET literal variable = literal value

where:

line number :

is any legal line number.

literal variable :

is any legal string variable name (A$, B$, C$)

literal value :

is any legal string variable name or any legal string constant.

Again, BASIC will evaluate the left hand side of the equal sign and store that value in the variable named on the right hand side of the equals sign. For example:

```
10  LET A$ = "CHARLIE'S ANGELS"
20  LET B$ = A$
```

Line number 10 will assign the literal string "CHARLIE'S ANGELS" to the literal variable called A$. Note that the double ampersand (quotes) marks are used only to define the string and are not actually stored with the string. Line number 20 will take the value stored in literal string A$ and proceed to store that value in literal string B$. B$ and A$ now contain the string: "CHARLIE'S ANGELS".

Examples of Valid LET Statements:

```
10  LET A = 0
20  LET P$ = "STAR TREK"
30  LET R$ = P$
```

In this version of BASIC, the word "LET" is optional and does not have to be specified when forming a LET statement.

```
40  T = 234
50  A(23) = 5
```

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

MAT statements:

It is often necessary to fill an array with several different values. If there are many values to be stored it is usually too time consuming to use LET statements. BASIC allows the user to fill arrays quickly and easily with MAT statements. The general form of the MAT statement is...

line number MAT array name value list

where...

line number :
is any legal line number.

array name :
is the name of the array to be filled followed by the starting location of the fill.

value list :
is the list of numeric constants separated by commas.

The MAT statement will fill the array named with the numbers in the value list starting at the element in the array desired. Examples of MAT statements are...

```
10 MAT A(1)  2,4,3,66,5,-7,.55
20 MAT B(1,1)  0,9,7,45,78,-999
30 MAT R(5,8)  23,34,45,56,67,78,90
```

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

POKE statement:

The POKE statement allows the user to put any integer value from zero to two-hundred fifty-five (0 - 255) into any single memory location or any value from two-hundred fifty-six to sixty-five-thousand-five-hundred-thirty-five (256 - 65535) into any two consecutive memory locations. The general form of the POKE statement is...

line number POKE location , value

where...

line number :
is any legal line number.

location :
is any integer value from 0 to 65535 in decimal.

value :
is any integer value from 0 to 65535 in decimal.

The following example...

```
10 POKE 1000 , 20
```

will place the value of twenty (hex value 14) into memory location 1000 (hex location 314).

One of the most important uses of the poke statement is to specify the number of digits of accuracy to the right of the decimal point theprogrammer wishes to have. This is done by poking the number of decimal digits needed into memory location 240. For example the following program...

```
10  POKE 240 , 2
20  PRINT 22/7
30  END
```

will print out...

```
3.14
```

```
10  POKE 240  , 4
20  PRINT 22/7
30  END
```

will print out...

3.1428

The number of digits to the right of the decimal point is set to zero each time the RUN or NEW command is used and when BASIC is first executed.

PRINT statements:

PRINT statements are used to output data to a list device such as a CRT or TTY. A PRINT statement may consist of literal or numeric values separated by commas (,), semi-colons (;) or colons (:). Literal values may be in the form of variable strings (such as A$,G$,K$) or constant strings enclosed in double quote marks (such as "PROGRAMMA CONSULTANTS", "DOCUMENTING IS DULL"). Numeric variables may be in the form of constants (such as 5,285,19.56), symbolic variable names (such as A,P6, A97)) or numeric expressions (such as 5*7, 34-R,P/A). If there are no values following the word PRINT a line-feed carriage return is generated only.

The destination of the values to be printed may be specified by the programmer. This is done by entering a pound sign (#) followed by a single digit zero (0) or one (1) at the beginning of a PRINT statement. If #0 is used print out will be to the CRT screen only. If a #1 is used the values to be printed will be output to the TTY only. If neither #0 or #1 is used the values will be printed to the CRT screen.

If more than one value is to be printed in a single PRINT statement the values must be separated by a comma, colon or semi-colon. If a colon (:) is used to separate two values in a PRINT statement BASIC will print the two values with no spacing between them. If a semi-colon (;) is used BASIC will place a single space character between the two values. If a comma (,) is used for separation between two values BASIC will divide the print device into 4 fields of 8 columns each and print the second value at the beginning of the next field. When all the values in the PRINT statement have been printed BASIC will move the cursor to the beginning of the next line. However, if a separator character appears at the end of the PRINT statement BASIC will carry out the action required for the separator and not move the cursor to the beginning of the next line. PRINT statements have the following general form...

line number PRINT list device print list

where...

line number :
is any legal line number.

list device :
is optional specifying where output is to be directed.

RELOAD statement:

The RELOAD statement is fully explained under the DUMP/RELOAD section of the commands. The user is instructed to check this section for a full explanation of the use of the RELOAD statement.

The RELOAD statement can be used under program control to restore the symbol table and dynamically allocated variables from a cassette or disk file. The general form of the RELOAD statement is as follows:

```
line number   RELOAD  XX
                      disk file name
```

where...

XX is any valid cassette file name previously DUMPed

disk file name is any valid disk file type D that was DUMPed

print list :
any number or combination of literal or numeric values separated by colons, semi-colons or commas.

Examples of Valid PRINT Statements:

```
10  PRINT "STAR TREK"
20  PRINT "SHIELDS CONTAIN";S
30  PRINT A$,A,B$,B
```

REM statements:

A very useful feature of FBX is the REM statement. A REM statement is used to document or comment on a BASIC program without affecting program execution. In other words, a REM statement does absolutely nothing except supply the user with information about a program. The general form of a REM statement is...

line number REM any text cr

where...

line number :
is any legal line number.

any text :
is any comment or message the user wishes to enter

Examples of Valid REM statements:

```
10  REM  THIS PROGRAM IS BY GEORGE WASHINGTON
225 REM  THIS PROGRAM COMPUTES TEMP. AT V. FORGE
664  REM
```

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

USER statement:

It is sometimes useful to be able to call a user written machine language program from a BASIC written program. These programs or subroutines may be in the PDS monitor or user written routines in RAM memory. This can be done by the USER statement. The USER statement has the following general form...

line number USER (machine language subroutine location)

where...

line number :
is any legal line number.

machine language subroutine location :
is the base ten location of the subroutine to be executed.

The subroutine must end with a RTS (39) instruction. This will return to the next line in the basic program being executed.

Before returning from the subroutine the user must set the Carry bit flag. This can be done with the SEC (OD). If the Carry bit flag is cleared when the RTS is reached BASIC will respond with...

ERROR 255 XXX

If the Carry bit flag is set when the RTS is reached BASIC program execution will continue with the next statement.

To inactivate the Carry bit flag error facility the program must...

POKE 5215 , 54856

sometime before the USER statement is reached.

To reactivate the Carry bit flag error facility the program must ...

POKE 5215 , 51710

Examples of Valid USER Statements:

250 USER (4321)
1903 USER (8000)

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

VII. FULL BASIC EXTENDED: COMMANDS

Version 2.0

CLOAD command:

The CLOAD command is used to enter a BASIC program or part of a BASIC program from the cassette unit into the computer workspace. The program need not be in completed form, however, it must be a BASIC source. The CLOAD command has the following general form...

CLOAD BASIC program name

where...

BASIC program name :
is a one or two character program name.

The BASIC program will be searched for on the tape and only the program with the correct name will be read in. If no name is specified then the first program found will be loaded. When the blinking cursor returns the program has been read in. Checksum and Trailer messages will be displayed if read errors occur.

CONVERT command:

Prior to Version 2.0 of FBX BASIC, all user written BASIC programs were stored in a form very similar to what is actually displayed upon the CRT screen, at the time that one does a LIST. Specifically, each BASIC line of source code contained the following internal form:

1. 2 Byte BASIC Statement Number in Binary
2. 1 Byte Line Length in Binary
3. Source BASIC Statement in ASCII with variable length

In order to implement new features easily and in order to speed the interpretation phase of BASIC, FBX 2.0 and above employees a technique called "tokenizing." This technique encodes all BASIC language keywords, such as LET, PRINT, IF..., with a one byte representative code. The changes to the above method of internal representation occur in the source BASIC statement itself. These changes are merely, an encoding scheme for all keywords. Therefore, it would be expected that the total memory requirements of a program will decrease. As a result of having all keywords encoded at the time that a BASIC statement is entered, then the BASIC Interpreter performs at a substancial increase in speed.

The purpose of the CONVERT command is to translate those BASIC source programs that were created for TBX Version 1.3 or below into the new internal form of FBX 2.0. To use this command, the user merely loads the old BASIC program from cassette or disk, and then issue the CONVERT command. As each line is being translated, it will be displayed upon the console. For example:

```
CLOAD  BA                LOAD PAYROLL
CONVERT                  CONVERT
CSAVE  BA                SAVE PAYROLL
```

CSAVE command:

The CSAVE command is used to save a BASIC program on to cassette storage. The BASIC program need not be in completed form to be saved. The general form of the CSAVE command is...

CSAVE BASIC program name

where...

BASIC program name :
is any one or two ASCII characters.

When the CSAVE command is executed and the cassette unit is set to record, the contents of the program workspace will be stored on to cassette. When the program is copied onto cassette, the blinking cursor will be displayed. The CSAVE command copies, it does not move the BASIC program onto cassette. The original program is still in memory. If errors occur during CSAVE, Checksum or Trailier message will be displayed.

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

CNTL A command:

If the user wishes to stop execution of a BASIC program, instead of resetting the computer, he may enter a CNTL A character followed by a carriage return during any INPUT statement (when the cursor is blinking). This will cause an error to be displayed on the particular input statement and a return to Command Mode.

CNTL D command:

The user may stop execution of the BASIC interpreter and give control to the PDS DEBUG routine. This is done by entering a CNTL D character whenever a blinking cursor is being displayed.

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

*** D I S K ***

CNTL E command:

The user may transfer control from the BASIC interpreter to the OS/1 system by entering a CNTL E character. The system will respond with a $ prompt. The user then may issue any legal OS/1 executive directive. This is for disk users only !

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

DUMP/RELOAD commands:

If at any time during program execution of a BASIC source program the user wishes to stop the program and continue at a later date, he may use the DUMP command to save the "state" of the program and the BASIC Interpreter, and later on he can issue the RELOAD command and continue with the program.

For example, assume that the user is playing a game of STAR TREK and he chooses to stop and continue the game later on. First, the user must stop execution of the program. This can be done by typing Cntl A during any INPUT request, or by typing SAVE when the game asks for the next command. The user then proceeds to enter the DUMP command.

The function of the DUMP command is to save to a cassette or to a disk file a copy of the symbol table used by the BASIC interpreter, and a copy of those variables that were dynamically allocated. The symbol table in BASIC is a listing of all of the variables that are used in a program together with their corresponding values. Variables that are dynamically allocated are those that are specified by using the DIMension statement. Note that the DUMP command never saves a copy of the actual BASIC source program itself.

The DUMP command has the following general form:

For Cassette: DUMP XX

For Disk: DUMP disk file name

where...

XX is any valid cassette file name.

disk file name is any disk file of type D for DATA.

When the user wishes to continue the game at a later time, he must do the reverse of the above. First, he must load the source BASIC code for the STAR TREK program back into the computer. Next, he must issue the RELOAD command with the proper cassette/disk file name , in order that the symbol table and the dynamic allocated variables can be restored. The general form of the RELOAD command is as follows:

For Cassette: RELOAD XX

For Disk: RELOAD disk file name

where:

XX is any valid cassette file name previously DUMPed.

disk file name is any disk file of type D that was DUMPed.

After having RELOADed a BASIC session, the user must type a GOTO command to restart program execution at the point of where the program is re-started. If the user tries to start the program with the RUN command, then all variables will be reset to zero. As an example, if the program was terminated at line 1035, then the user would re-start the program by issuing a GOTO 1035.

The following is an example of DUMPing and RELOADing a game of STAR TREK...

```
34 UNITS HIT SECTOR (5,2)  1920 UNITS REMAINING
COMMAND?  SAVE
DUMP  S1                   DUMP TREKSAVE
.
.
.
CLOAD  ST                  LOAD STARTREK
RELOAD S1                  RELOAD TREKSAVE
GOTO 1035
COMMAND?  PHASERS
```

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

LIST command:

The LIST command is used to direct a copy of the current program in memory to the CRT or TTY. This is useful for debugging because it allows the user to see exactly what he has typed in so far. The general form of the LIST command is...

LIST device designator starting address ending address

where...

device designator :
is optional and defaults to #0 causing program listing to be directed to the CRT screen, #1 causes all listing to be directed to the printer and CRT screen.

starting address :
if no ending address is given this line number will be listed, if an ending address is given all lines between starting and ending will be listed.

ending address :
the last line of the group from starting to ending address.

The LIST command can be used with any, all or none of the attributes above. LIST alone will cause the entire BASIC program to be listed in pages. Once a full screen has been printed or viewed a carriage return will cause the next page to be listed. To get out of LIST mode before all pages have been called, type escape.

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

*** D I S K ***

LOAD command:

The LOAD command is used to read a BASIC program off of disk storage and into the BASIC program workspace. The LOAD command has the following general form...

LOAD file name

where...

file name :
is any BASIC file name that has previously opened on disk storage. This pertains to file type B only.

RENUMBER command:

The purpose of the RENUMBER command is to re-sequence the line numbers of a BASIC source program into ascending sequence. The way that this is accomplished is to assign the first statement encountered, line number 10. Each line of source code thereafter is automatically incremented by 10 and assigned that value. This is all done automatically by the BASIC program. GOTO statements and GOSUB statements are automatically re-organized to proceed to their new line numbers.

The user may enter the RENUMBER command at any point in time that he feels that he wishes to reorganize the line numbers of his program. For example:

```
RENUMBER
```

NEW command:

The NEW command is used to clear the program workspace. When the user has finished with a program in the computer and wishes to write another one, he uses the NEW command. This will erase the program that is presently residing in the computer and resets all variables and internal flags. It will not erase programs that are on disk or cassette.

The NEW command has the following general form...

NEW

No line number is allowed.

RUN command:

Once a set of instructions have been entered into the computer and debugged, it is time to try it out. This is done by using the RUN command. The RUN command instructs the computer to immediately set all variables to zero, clear out all arrays and strings and begin program execution starting at the first line in the program. The RUN command has the following general form...

RUN

No line number is needed.

*** D I S K ***

SAVE command:

The SAVE command is used to save BASIC programs onto disk storage. The file on disk must have been previously opened thru the OS/1 EDITOR. The program need not be completely written and debugged to be saved. BASIC will save whatever has been entered into the workspace so far. The save command has the following general form...

SAVE file name

where...

file name :
is any previously opened file name on disk storage type B.

SYSTEM NAME	SYSTEM NUMBER	CATALOGUE NUMBER
PROGRAM NAME	PROGRAM NUMBER	DATE DOCUMENTED

SIZE command:

The SIZE command may be used when the user wishes to know how much memory he has remaining. The general form of the SIZE command is...

SIZE

When the SIZE command is used, the computer will respond by printing out the number of bytes of unused memory. The number printed is in base 10.

VIII. APPENDICES

A. Error Codes for FBX 2.0

B. Cassette Loading Procedure

C. BASIC language Quick Reference

1. Statements
2. Commands
3. Functions

D. Glossary

APPENDIX A

Error Codes

Error number	Description of error
0	CNTL A entered during execution of INPUT statement.
1	Statement is not part of BASIC statement set.
2	Tape read error, reload tape.
3	Illegal character in the statement or inputed data.
4	Missing closing quote in string constant.
5	Arithmetic expression too complex.
6	Illegal arithmetic expression.
7	Label assigned in GOTO or GOSUB does not exist.
8	Division by zero attempted.
9	Subroutines nested too deep.
10	RETURN with no prior GOSUB.
11	Illegal variable name.
12	Statement is not recognized.
13	Error in use of parenthesis.
14	Memory error.
15	unassigned to date
16	Illegal FOR variable
254	Carry bit not set in USER function

APPENDIX B

Cassette Loading Procedure

1. Enter Sphere DEBUG routine,
2. Open (CNTL O) location 0009 h, change (space) to 00,
3. Open and change location 0033 h to 00,
4. Open and change location 0034 h to 00,
5. Open and change location 0038 h to F0,
6. Open and change location 0039 h to 60 for CASS II
   50 for CASS I
7. Open and change location 003A h to 01,
8. Open and change location 003B h to 01,
9. Open and change location 003C h to the first byte of the two byte location where the program is to be stored,
10. Open and change location 003D h to the second byte of the storage location,
11. Open and change location 003E h to 00,
12. Open and change location 003F h to 00,
13. Open location FB00 and type CNTL J (jump),
14. Open location FB91 and type CNTL J.

locations 33 and 34 can be set to the program name in hex if it is necessary to search for a specific program. For example:
0033 42
0034 41
would cause the tape to be read until BA was found and read in.

APPENDIX C

BASIC Quick Reference

Statements

*CLEAR

clears CRT screen, HOMES cursor.
line number CLEAR

DIM

dimensions strings or arrays.
line number DIM symbolic name (first dimension , second dimension)

END

terminates BASIC program.
line number END

* FOR

begin of FOR ... NEXT loop.
line number FOR index variable = lower limit TO upper limit

GOSUB

calls BASIC subroutines.
line number GOSUB subroutine location

* GOTO

unconditional transfer.
line number GOTO destination

* IF

conditional transfer.
line number IF expression relation operator expression statement

* LET

assigns a value to a variable.
line number LET symbolic name = expression

*MAT

assigns values to an array
line number MAT array name = constant list

* indicates statement may be used with or without line number.

NEXT

end of FOR ... NEXT loop.
line number NEXT index variable

* POKE

pokes numeric value into one or two memory locations.
line number POKE(location , value)

* PRINT

outputs values to list device.
line number PRINT device designator print list

* RETURN

returns from BASIC subroutine program.
line number RETURN

* indicates statement may be used with or without line number.

Commands

CLOAD

loads program from cassette storage.
CLOAD cassette file name

CNTL A

stops execution of BASIC program returns user to DEMAND mode.
control and A keys pressed at the same time

CNTL D

returns user to PDS DEBUG.
control and D key pressed at the same time

CNTL E

returns user to OS/1 system nucleus.
control and E key pressed at the same time

CSAVE

saves program onto cassette storage.
CSAVE program file name

DUMP

dumps symbol table and variable list to disk file type D.
DUMP disk file name or XX

EDIT

edits line of program.
EDIT line number

LIST

lists contents of program workspace to list device.
LIST device designator starting line number
, ending line number

LOAD

loads program from disk file.
LOAD disk file name

NEW

clears program workspace and variables.
NEW

RELOAD

restores variables and symbol table previously DUMPed.

RELOAD XX or File-Name

RUN

starts execution of BASIC program, clears all variables.

RUN

SAVE

saves BASIC program onto disk file.

SAVE disk file name

SIZE

returns number of remaining bytes of memory.

SIZE

RENUMBER

allows the BASIC source statements of a program to be numerically arranged in ascending sequence, beginning with the first statement as 0010, the next as 0020, and incrementing each statement thereafter by ten.

RENUMBER

CONVERT

allows the BASIC source statements for a Version 1.3 TBX to be converted to the new FBX 2.0 format.

CONVERT

Functions

ABS

returns absolute value of an expression.
ABS(numeric expression)

INT

returns integer portion of a numeric expression
truncates decimal part.
INT(numeric expression)

LEN

returns number of characters stored in string.
LEN(string name)

PEEK

returns value stored in memory location.
PEEK memory location

RND

returns a random number between 0 and 1
RND (0)

SGN

returns sign of expression; 1 if positive, -1 if negative
or 0 if zero.
SGN(numeric expression)

STR

returns sub string of string.
STR(string name , beginning position , length)

PI

returns value of pi 3.14159
PI

APPENDICES D

GLOSSARY

Algorithm
A specific method designed to yield a solution to a problem.

Alphanumeric
The set of ASCII characters which include the letters A to Z, the digits Ø to 9 and the special characters.,;: = ?!/* -@$#%&()

Attribute
A rule or characteristic which applies to a statement, instruction, command or item of data

Argument
A numeric or literal constant or variable which is operated on by a predefined function. X is said to be the argument of the sine function Z=SIN (X)

Array
A set of numeric variables that are associated with a single symbolic name. See also vector matrix

ASCII
American Standard Code of Information Interchange

BASIC
Beginners All-purpose Symbolic Interpretive Code

BASUC Command
An instruction to the BASIC compiler entered without a line number, i.e. LIST, RUN etc.

Binary
A number system in base two

Branching, Conditional
Transfer of program control which is dependent upon the outcome of a logic test

Compiler
A program usually written in Assembler, which converts a high level language into object code

Computer
A network of logic circuits, memory and I/O circuits capable of addition and storage of data received

Contiguous
In a logical, usually arithmetic order

Constant
An amount or number which does not change

Constant, Literal
A set of ASCII characters enclosed in quotation (") marks

Constant, Numeric
A series of digits (Ø-9)

CPU
Central Processing Unit

CRT
Cathode Ray Tube

Data
Literal or Numeric variables supplied to the computer for storage

Debug
A M68ØØ subroutine on V3N PROM set
To remove errors from a program

Decimal
A system of computation using base 10 numbers i.e. 5, 10,1000

Disk
A mass storage device used for hi-speed data transfers

Diskette
A flexible magnetic card which is read from or written upon in a disk

Edit
A M6800 subroutine on V3N PROM set

Element
A single numeric storage location in an array

Equation
A target variable followed by an equal sign, followed by an arithmetic expression.

Expression
A series of variables and/or constants combined with arithmetic operators.

Flag
A variable used to test for a condition usually Ø or 1

Flowchart
A series of graphic shapes used to document or draft a program. See-Appendix, Flowchart

Function
A predefined expression available to the programmer

Hardware
Electrical devices used by the computer for storage or communications

Hexadecimal
A system of numbers using base 16

Index Variable
The variable in a For/Next loop

Input
To answer questions or provide data to the program

Interface
To provide the CPU with a means of communications with external devices

Instruction
A word or phrase used to communicate with the compiler

I/O
Input/Output

Loop
A group of instructions which are executed more than once in succession

Matrix
An array with 2 subscripts

Memory
A device used by the computer to store data or program instructions

Numeric
Having a number or amount as a value. No letters or special characters

Octal
A system of numbers based on digits 0 through 7

Object Code
Low level language understood by the computer

Operator
The mathematical signs +,-,*,1,(,), .

Order of Operation
A logical sequence of evaluation of expressions. Order is:
1. sign of number
2. multiplication/division
3. addition/subtraction

Output
A response by the computer of data received and acted upon

Overflow
To go beyond the limits on the number range

Parameter
An attribute or limitation of a function or expression

Program
A series or group of instructions which cause the computer to act on data

PROM
Programmable Read Only Memory

RAM
Random Access Memory

Range, Line Numbers
Line numbers must be integer values such that 0 LN 65536

ROM
Read Only Memory

Routine
A group or subject of instructions which accomplish a specific goal

Scalar
A numeric variable with only one element associated with it

Software
Instructions given the computer to accomplish a task

Statement
A BASIC instruction preceded by a line number

String
A series of Alpha numeric values either a constant or assigned to a literal variable

Subroutine
See - Routine

Subscript
An integer value describing a location in an array

Symbolic Name
For Scalar Numeric Variables: a single letter (A toZ) followed by an optional digit (0 to 9)
For Arrays: a single letter
For literal variables: a single letter followed by the $ character

Syntax
The rules of grammar for a computer language

TTY
Teletype I/O device

Unary
One or single. The minus plus sign preceding a constant or variable is the unary operator

Underflow
To go under the limit of the range of numbers

Variable
That which changes or can be changed

Variable, Literal
A symbolic name used to identify the location of a literal string in memory

Variable, Numeric

A symbolic name used to identify a numeric value in memory

Vector

An array with 1 subscript