FAST BASIC I ROM
FOR THE HEWLETT-PACKARD 9830A/B COMPUTER

INSTRUCTION MANUAL

Infotek Systems

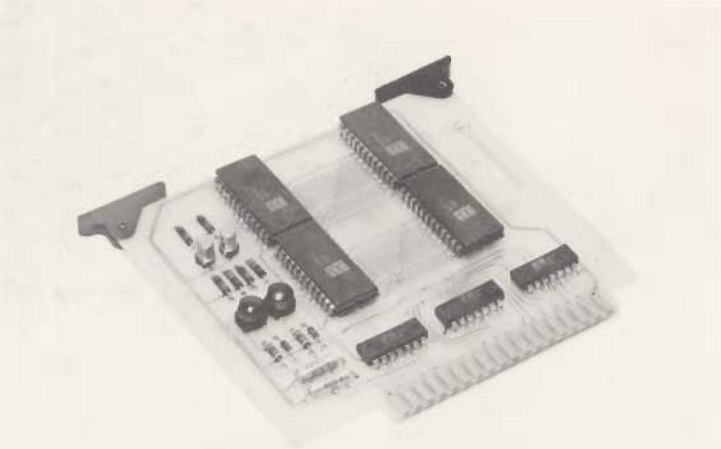
1400 N. Baxter St. • Anaheim, Calif. 92806 • (714) 956-9300 • TWX 910-591-2711

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

INFOTEK FAST BASIC I ROM

FAST BASIC I ROM
(EXTERNAL)



FAST BASIC I ROM
CIRCUIT CARD
(INTERNAL)

INSTRUCTION MANUAL

INFOTEK FAST BASIC I ROM FOR THE HEWLETT-PACKARD 9830 A/B* DESK-TOP COMPUTER

HP 9830A/B with the Infotek FD-30 Mass Memory

*a product of Hewlett-Packard Company

Infotek Systems

1400 N. BAXTER ST. • ANAHEIM, CALIF. 92806 • (714) 956-9300 • TWX 910-591-2711

TABLE OF CONTENTS

INTRODUCTION

The FAST BASIC I ROM is a general purpose expansion of the HP 9830A/B basic language. With its 27 functions, statements and commands, FAST BASIC I is certainly the most powerful ROM ever provided for the 9830A/B. FAST BASIC I brings the language of your 9830 to a level comparable to large time share systems.

The new array manipulation related statements allow moving all or portions of arrays from one to another or from one part of an array to another. With a single statement, arrays can be searched based on any of four relational operators. Arrays can be compared or swapped, even if their dimensions are dissimilar. The average speed of these array-related operations is over 10,000 elements per second.

Substantial power and speed improvement is also provided to the 9880B Mass Memory System user. FAST BASIC I permits operating directly on data item pointers. It permits updating a single or group of items without the necessity to read and rewrite an entire record up to the point where an update is desired. A significant increase in mass memory speed may be obtained when using these two new capabilities. Another powerful mass memory function allows using both units simultaneously. In such cases, redefining file assignments is unnecessary. This feature saves substantial time as directory seeks are eliminated. Other new mass memory related functions allow interrogating the file size, the record number currently in use, the number of available words remaining in a record, and the status of the mass memory system.

Eighteen additional statements, functions and commands further enhance the general utility and ease of programming the 9830A/B. One of the most powerful of these is SWAP. Parameter passing, a sorely needed capability, is now possible and convenient with SWAP.

Effective incorporation of FAST BASIC I can provide extraordinary increase in program execution speed. Many operations previously not feasible or possible can now be executed with speed and convenience.

SYNTAX CONVENTIONS

brackets [ ] — items enclosed in brackets are optional.
coloring — items printed in color must appear as shown.

INSTALLATION

Installing the internal ROM:

1. Place input power switch in the OFF position.

2. Remove the input power cord from the wall outlet and the power jack at the rear of the HP 9830A/B.

3. Lift the thermal printer from the computer (if so equipped) and place to one side.

4. Remove the six screws from the top cover of the computer (Figure 1).

5. Slide the top cover back about two-thirds of the way by using the plastic handles at the back of the cover.

6. Remove the single screw that retains the two crossed aluminum hold-down brackets (Figure 2). Note the location of the brackets and how they are attached. Remove the brackets.

7. The first three card positions behind the front panel along the left side of the computer are reserved for internal optional ROM's. Locate any ROM slot from among the three specified. The slot will have a black card guide on the left side and a red guide on the right side. Figure 3 shows the location of the first optional ROM location.

8. Position the circuit card over the card guide with the component side of the card facing toward the rear of the computer. Confirm that the guide and handle colors match.

9. Carefully lower the circuit card down the guides and into the connector well until contact is made with the connector. Be certain that the edges of the card are within the edges of the connector well.

10. Apply even pressure with the thumbs to the top of the handles to seat the circuit card in the connector. The card is fully seated when the top is approximately even with the cards in front or in back.

11. Replace the two aluminum hold-down brackets and secure with one screw.

12. Slide the cover forward and secure with the six screws.

13. Replace the thermal printer.

14. Verify that the input switch is in the OFF position.

15. Connect the power cord to the computer and the wall outlet. This completes the installation.

CAUTION

When power is first applied to the computer after installation of the FAST BASIC I ROM, watch for the lazy T on the display. If it does not appear within a few seconds after the power switch is placed in the ON position, immediately place the switch in the OFF position and contact Infotek or your Infotek representative.

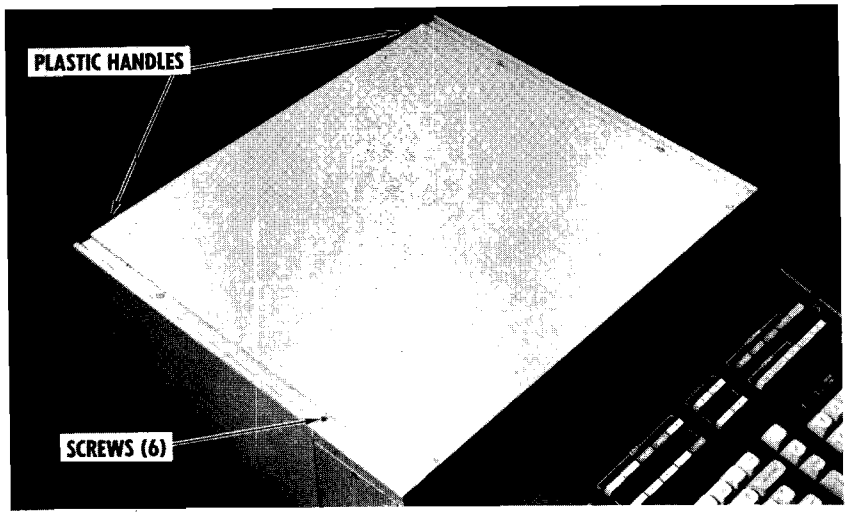


Figure 1.
Removal of Cover

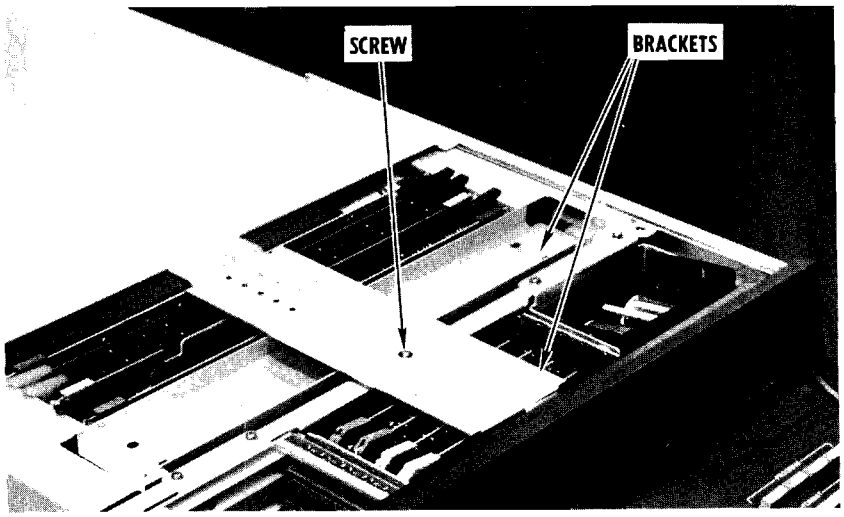


Figure 2.
Removal of Brackets

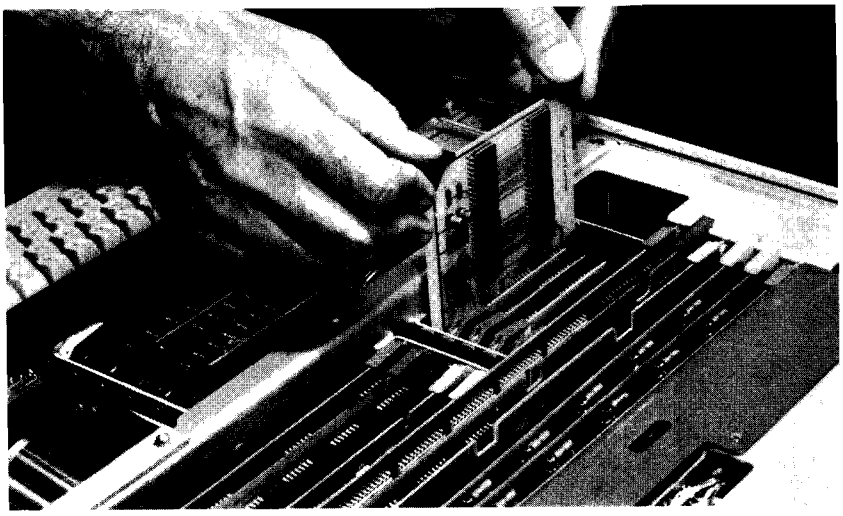


Figure 3.
Location of
FAST BASIC I ROM
Circuit Card

NOTE: The FAST BASIC I ROM may occupy any of the first three slots behind the front panel.

SEND STATEMENT

The SEND statement is used to move data between locations in arrays. SEND may be used to transfer data from one array to another, or from one area in an array to another area in the same array.

Since SEND moves sequential locations in memory, an understanding of how arrays are stored in the computer is desirable. An explanation of this aspect of the computer's internal operation may be found in Appendix B.

SYNTAX:

SEND array name (subscripts) **TO** array name (subscripts)

EXAMPLE:

SEND A(R1,C1) TO B(R2,C2)

Array elements are sent from source array A starting at row R1, column C1, to the destination array B starting at row R2, column C2. The SEND operation continues sequentially until the last element of either array is reached. Arrays A and B need not have the same dimensions, nor the same number of dimensions. However, the arrays **MUST** be of the same precision. A and B may be in the same array. Subscripts **MUST** be provided, even when an entire array is to be moved.

USAGE EXAMPLES:

SEND is a much faster replacement for the MAT= statement of the MATRIX ROM.

FAST BASIC I

```
10 DIM AI[50,50],BI[50,50]
15 LOAD DATA 3,A
20 SEND A[1,1] TO B[1,1]
```

MATRIX METHOD

```
10 DIM AI[50,50],BI[50,50]
15 LOAD DATA 3,A
20 MAT B=A
```

Both methods transfer information from array A to array B. SEND is about 200 times faster on integer arrays. Because the dimensions of the source and destination arrays in the SEND statement need not be identical, SEND will function where MAT= **WILL NOT.**

```
10 DIM A[10,10],B[100]
80 SEND A[1,1] TO B[1]
100 SEND A[6,1] TO B[51]
140 REDIM A[5,10]
150 SEND A[4,4] TO B[1]
```

SEND STATEMENT [Cont.]

Line 80 transfers each element in array A to the corresponding element in vector B on a row-by-row basis. Line 100 shows how SEND can be used to move a part of an array to a dissimilarly dimensioned array with simultaneous termination. Lines 140 and 150 show the ability to move an arbitrary number of locations. This illustrates the ability of SEND to operate on arrays with inconsistent dimensions. The following program reads data into an array and then multiplies each row of the array by a constant read from a data statement.

```
10 DIM A[10,10],T[10]
20 DATA 1,2,3,4,5,6,7,8,9,10
110 LOAD  DATA 1
120 FOR I=1 TO 10
130 SEND A[I,1] TO T[1]
140 READ M
150 MAT T=(M)*T
160 SEND T[1] TO A[I,1]
170 NEXT I
```

The SEND statements in lines 130 and 160 transfer rows of array A to and from the temporary array T. By using only one FOR-NEXT loop and limited subscripting, a decrease in the amount of time required to run the program may be expected. The **PERCENT** time saved is directly proportional to the array dimensions. Because the source and destination arrays used in SEND may be the same array, the SEND statement is ideal for initializing arrays. It is particularly useful when one or zero is not desired as a constant. As explained in Appendix B, SEND can move four words at a time, provided that the number of words to be moved is an integral multiple of four. Accordingly, in the following example, the first four words of the array must be pre-initialized before SEND is used. The following statements illustrate the use of SEND for initialization.

```
10 DIM A[30,30],BS[30,30],CI[30,30]
120 A[1,1]=6.02E+23
130 SEND A[1,1] TO A[1,2]
150 B[1,1]=B[1,2]=4.8
160 SEND B[1,1] TO B[1,3]
180 C[1,1]=C[1,2]=C[1,3]=C[1,4]=6
190 SEND C[1,1] TO C[1,5]
```

Because each of the arrays has an **INTEGRAL MULTIPLE OF FOUR WORDS,** destinations specified in the SEND statements cause the second group of four words to be the first destination as indicated in lines 160 and 190. SEND can initialize an array to any value. In lines 180 and 190, each element of array C is set to 6. This could be done with matrix ROM instructions, but the sequence required would be:

```
10 DIM AI[30,30]
100 MAT A=CON
110 MAT A=(6)*A
```

The MAT scaler will take over two hundred times longer to execute for a 900-word integer array.

COMPARE STATEMENT

The COMPARE statement tests for equality between each word of one array and the corresponding word of another array. The comparison continues until the last word of both arrays has been tested. The return variable of the statement then returns the number of inequalities found. COMPARE operates on arrays of any precision. The dimensions need not be consistent; however, both arrays must be of identical precision and contain an equal number of words from the beginning subscripts to the end of the arrays. COMPARE executes hundreds of times faster than any previous BASIC language sequence that could perform the same function. The percent increase in speed is directly proportional to array dimensions. A pair of 6,000 word arrays will be compared 500 times faster with COMPARE than with a BASIC program.

SYNTAX:

COMPARE array name (subscripts) **TO** array name (subscripts), return variable

EXAMPLE:

COMPARE A(R1,C1) TO B(R2,C2), N

okgem
4/31/1981

This statement will compare the contents of array A starting at row R1, column C1, to the contents of array B starting at row R2, column C2. The comparison continues sequentially to the end of the arrays (see Appendix B). The number of inequalities is returned in N. The arrays A and B must contain the same number of words from the referenced starting location to the end of the arrays. Array subscripts must be used.

USAGE EXAMPLES:

One use of the COMPARE statement is in business systems. It can be used to compare present data with data from some previous period to see if there have been any changes. Array C contains current period data and L contains a previous period.

```
340 COMPARE C(1) TO L(1),N
350 PRINT N"CHANGES IN INVENTORY THIS PERIOD."
```

Line 340 returns the number of inequalities in N and line 350 prints the result of the comparison.

SWAP STATEMENT

The SWAP statement exchanges the contents of two arrays, simple variables, or strings. SWAP is over ten thousand times faster than any method previously possible. Moreover, SWAP can be used for passing arrays or strings as parameters in sub-routines, a powerful programming tool which was previously unavailable.

SYNTAX:

SWAP { string name / variable name / array name } , { string name / variable name / array name }

EXAMPLE:

OK BCM 5/31/1981

```
SWAP A,B
```

This statement exchanges the contents of array A with the contents of array B. The statement may also be used to swap strings if the string variables ROM is installed.

```
SWAP F$,G$
```

will exchange F$ with G$. SWAP may also be used to exchange simple variables as follows:

```
SWAP A1,B2
```

SWAP may be used on simple variables only if their names contain a digit. The SWAP statement assumes that a letter without a digit following is an array name.

Arrays used in a SWAP statement must have the same number of dimensions and must be of the same precision. The dimensions and the number of elements may differ. SWAP will not transpose array elements or sub-strings.

SWAP STATEMENT [Cont.]

The following example uses SWAP to permit the use of the same program segment for sorting three different arrays.

```
 10 DIM A[10],B[10],C[10]
180 GOSUB 310
200 SWAP A,B
210 GOSUB 310
230 SWAP A,B
250 SWAP A,C
260 GOSUB 310
280 SWAP A,C
300 END
310 REM THIS IS THE  SORT SUBROUTINE
320 FOR I=1 TO 10
330 FOR J=I+1 TO 10
340 IF A[J] >= A[I] THEN 380
350 T=A[J]
360 A[J]=A[I]
370 A[I]=T
380 NEXT J
390 NEXT I
400 RETURN
```

Lines 320 thru 400 are the sorting sub-routine. Line 180 starts the process on array A. On return to line 200, arrays A and B are transposed and line 210 initiates the SORT on the contents of array B which have been "passed" to array A. At line 230, the sorted contents of array B are passed back to B. On line 250, the process repeats for array C. At line 280, all three arrays are sorted and the contents of each are again the original data.

SCAN STATEMENT

The SCAN statement is used to search an integer precision array to determine which array element best satisfies a specified relation. The relation may be greater than, less than, equal to, or not equal to. Additionally, the search may be done under an optional binary mask. SCAN may be used to locate maxima and minima in arrays, find which data item in a list of measured values comes closest to an expected value, or which item is or is not equal to a match variable.

SCAN is approximately 200 times faster than previously available BASIC language routines for a search using an equal-to or not-equal-to relation, with or without a mask. Additionally, it is ten times faster than the SEARCH command of the Advanced Programming II ROM, which is limited to a vector and operates only on the "equals" relationship.

SYNTAX:

SCAN array name $\left\{\begin{matrix}>\\<\\=\\\neq\end{matrix}\right\}$ match expression, return variable [, mask expression]

EXAMPLES:

okbpm
5/31/1981

1. SCAN A>X,L
2. SCAN A<X,L
3. SCAN A=X,L
4. SCAN A≠X,L
5. SCAN A=32,L,255

Where A is the name of an integer-precision array, X is any expression which reduces to an integer value, and L is the variable in which the location of the element which satisfies the relation is returned. Example 1 above finds the smallest element in A which is greater than X, while example 2 finds the largest elements in A which is smaller than X. The equal and not-equal relationships in examples 3 and 4 find the first element in the array that satisfies the relation. If the relation cannot be satisfied, a zero is returned.

SCAN may also be used to search an array under a binary mask. This is done by specifying a mask expression following the return variable as shown in example 5. This will search the least significant eight bits of every element in A to see if it contains an ASCII "space" (decimal code 32). The mask expression must reduce to an integer. To SCAN the most significant eight bits, the mask expression must reduce to -256.

USAGE EXAMPLES:

One of the most useful applications of SCAN is in finding maxima and minima of arrays. This is done by specifying the largest and smallest integers, respectively, with match expressions as follows:

SCAN STATEMENT [Cont.]

```
190 SCAN S<32767,M
200 PRINT "MAXIMUM OF S IS "S[M]
220 SCAN S>-32767,L
230 PRINT "MINIMUM OF S IS "S[L]
```

SCAN may be used to return other statistical information.

Continuing the above example:

```
240 M1=(S[M]+S[L])/2
250 SCAN S=M1,S1
260 IF S1=0 THEN 290
270 M1=S[S1]
280 GOTO 340
290 SCAN S>M1,L1
300 SCAN S<M1,L2
310 M2=S[L1]*(ABS(S[L1]-M1)<ABS(S[L2]-M1))
320 M2=M2+S[L2]*(ABS(S[L1]-M1)>ABS(S[L2]-M1))
330 SCAN S=M2,S1
340 PRINT "MEDIAN OF S IS ="M1
350 PRINT "ELEMENT OF S CLOSEST TO MEDIAN IS"
360 PRINT "ELEMENT"S1"WHICH IS"S[S1]"WHOSE DIFFERENCE"
370 PRINT "FROM THE MEDIAN IS"M1-S[S1]
380 END
```

The SCAN statement in line 250 attempts to locate an element in S which is equal to the median of S. If such an element exists, its location is returned in S1. However, if no element in S is equal to M1, SCAN returns a zero, and the SCAN statements in lines 290, 300, and 330, locate the element of S which is closest to the median, M1.

The value returned by SCAN is the sequential location of the returned element, relative to the beginning of the array. To change this number into subscripts for a two-dimensional array, it is necessary to convert the element number. For an array dimensioned A(I,J), the subscripts I1,J1, required to locate element N are obtainable from the following algorithm:

```
180 I1=INT((N-1)/J)+1
190 J1=(((N-1)/J-INT((N-1)/J)))*J+1
```

ROW AND COL FUNCTIONS

The two functions called ROW and COL take an array name as an argument and return the CURRENT WORKING DIMENSIONS of the array. Normally, the functions return the dimensions specified in the DIM statement, in which the array was defined. However, if the arrays have been redimensioned via the REDIM statement of the Matrix ROM, ROW and COL return the dimensions as specified in the most recently executed REDIM statement or implied REDIM.

SYNTAX:

Ok stm
5/31/1981

ROW(array name)
COL(array name)

USAGE EXAMPLES

These two functions are most useful when arrays are dynamically redimensioned with bounds specified by expressions. By using ROW and COL, arrays of variable size may be processed more easily:

```
100 DIM A[30,30]
  .
180 READ X,Y
190 REDIM A[X,Y]
200 MAT READ A[X,Y]
  .
490 GOSUB 1780
  .
1780 FOR I=1 TO ROW(A)
1790 FOR J=1 TO COL(A)
  .
```

Line 200 does an implied REDIM; whereas, line 190 does an explicit REDIM. Either of these methods is sufficient by itself.

TRIG FUNCTION

TRIG is a function which returns a **0** if the computer mode is DEGREES, a 1 if it is RADIANS, and a 2 if the mode is GRADS. It requires a dummy argument (an argument which has no effect on the function).

SYNTAX:

TRIG(dummy argument) Okmem 5/3/1981

USAGE EXAMPLE:

```
DISP TRIG(0)
```

MOD FUNCTION

The MOD function provides the integer remainder produced by dividing the first of its two arguments by the second. MOD(A,B) is the same as A-INT(A/B)*B. The Extended I/O ROM must be installed to use this function. Both arguments must evaluate to integer values between -32767 and +32767.

SYNTAX:

MOD(expression 1 , expression 2)

okmem
5/31/1981

USAGE EXAMPLE:

```
100 IF  NOT MOD(N,2) THEN 200
```

The above statement branches to line 200 if N is an even number. The MOD and COL functions may be used to calculate subscripts corresponding to a sequential location in a two-dimensioned array. For array A, the subscripts I,J corresponding to location N are:

```
10 I=INT((N-1)/COL(A))+1
20 J=MOD(N-1,COL(A))+1
```

Note that this is simpler than the example shown on page 10 which does not use MOD and COL.

UND FUNCTION

The UND function tests to see whether its argument is undefined. If undefined, UND returns a 1. If defined, the function returns a 0. UND may be used with either simple variables or array elements. Entire arrays or strings may not be used as arguments for UND. ERROR 40 (undefined variable) can be avoided by using the UND function.

SYNTAX:

UND(variable or array element)

USAGE EXAMPLE:

```
180 IF NOT UND(Q) THEN 210
190 DISP "VALUE FOR Q";
200 INPUT Q
210
```

Line 180 tests and branches around the input sequence, lines 190 and 200, if Q is defined.

DEC FUNCTION

The DEC function will change a decimal integer precision number to the equivalent octal number. It is the inverse of the OCT (octal) function. The range of numbers that may be converted by the DEC function is −32768 to +32767. Any number outside these bounds will return 100,000 (overflow).

SYNTAX:

DEC(expression) ok 3 p-m 5/3/1981

USAGE EXAMPLE:

```
120 PRINT "DECIMAL", OCTAL","DECIMAL","OCTAL"
130 PRINT
140 FOR I=6 TO 12
150 PRINT I,DEC(I),I+50,DEC(I+50)
160 NEXT I
170 END
RUN
DECIMAL         OCTAL           DECIMAL         OCTAL

 6               6               56              70
 7               7               57              71
 8               10              58              72
 9               11              59              73
 10              12              60              74
 11              13              61              75
 12              14              62              76
```

EXOR FUNCTION

The EXOR function performs a logical (boolean) exclusive-or operation on its two arguments. The function first converts the arguments to 16-bit integer words, then performs an exclusive-or and returns the result. This function is particularly useful in conjunction with the BIAND, INOR, and ROT operations provided by the HP Extended I/O ROM. EXOR requires two arguments which evaluate to integers between −32767 and +32767. The Extended I/O ROM must be installed to use this function.

SYNTAX:

EXOR(expression 1, expression 2)

ok rem 5/3/1981

USAGE EXAMPLE:

```
10 PRINT "A","B","EXOR(A,B)"
20 PRINT 3,1,EXOR(3,1)
30 PRINT 256,-1,EXOR(256,-1)
40 PRINT -1,-1,EXOR(-1,-1)
50 PRINT 0,0,EXOR(0,0)
60 END

RUN
A             B             EXOR(A,B)
 3             1             2
 256          -1            -257
-1            -1             0
 0             0             0
```

This programme worked as shown above, BCM 5/3/1981

OCT FUNCTION

The OCT function will change a base 8 (octal) integer precision number to the equivalent base 10 (decimal) number. It is the inverse of the DEC function. The range of numbers that may be converted by the OCT function is 100000_8 to 077777_8. Any number outside these bounds will cause an ERROR 4 to be issued. Octal numbers are limited to the digits 0 through 7.

SYNTAX:

OCT(expression)

ok rrm 5/3/1981

USAGE EXAMPLE:

```
100 DISP OCT(31)
     25
```

Causes the decimal equivalent, 25, to be displayed. To convert back to octal execute DEC(25). All arithmetic in the HP9830 is done on decimal values, so any octal numbers entered into a program must be converted to decimal before they can be used in computations.

FRAC FUNCTION

The FRAC function will return the fractional portion of a numeric expression as the result. The result is the equivalent of: expression - INT(expression).

SYNTAX:

FRAC(expression)

OK Bem 5/31/1981

USAGE EXAMPLE:

```
180 DISP "ENTER PART #";
190 INPUT P
200 IF FRAC(P) THEN 180
```

Note that the FRAC of negative values returns the difference between the argument and greatest integer that is less than the argument.

FILEID STATEMENT

FAST BASIC I

The FILEID statement provides the user with a method for determining tape cassette or floppy disk file parameters from within a running program. File number, size, and type, along with all other file header data, may be read into an array.

The information from the file header is returned as follows:

ARRAY WORD	CONTENTS
1	File Number
2	Current file size in words
3	File type
4	Absolute file size
5	Data type or first program line number
6	Last program line number
7	Common area in words

Note that this order differs from the TLIST command.

SYNTAX:

FILEID [#select code,] array name (subscripts)

EXAMPLE:

FILEID #5, A(N,1)

Reads the contents of the next file ID on a peripheral (select code 5) cassette or floppy disk into an array starting at location N,1. The information may be used by the program to test any parameter of the file header.

USAGE EXAMPLE:

```
10 DIM A[7]
20 FORMAT F2.0
30 PRINT
40 FORMAT "FILE  FILE  PHYSICAL  LOGICAL   FIRST"
50 WRITE (15,40)"    LAST   WORDS IN      FILE"
60 FORMAT " NO.  TYPE   LENGTH   LENGTH   LINE"
70 WRITE (15,60)"    LINE    COMMON     DESCRIPTION"
80 FORMAT "====  ====  =======  =======   =====    ====   ========"
90 WRITE (15,80)"  ========================="
100 PRINT
110 FILEID A[1]
120 WRITE (15,130)A[1],A[3],A[4],A[2],A[5],A[6],A[7];
130 FORMAT F3.0,3X,F5.0,4X,F6.0,3X,F6.0,2X,F6.0,2X,F6.0,5X,F6.0,2X
140 WRITE (15,20)" . . . . . . . . . . . . ."
150 PRINT
160 GOTO 110
170 END
```

FILEID STATEMENT [Cont.]

The file header information may be printed in a more elegant format than the TLIST command. This is particularly useful when the Infotek 16K memory is used. As the system TLIST provides four digits for file size and the memory permits five-digit file sizes the system TLIST would for example, show a 13,000 word file to have only 1300 words. The FILEID statement may also be used to advance the tape to the following file, inasmuch as the statement causes the computer to read the next file ID on the tape regardless of the current position of the tape.

A typical printing of a TLIST produced by the example utility program follows:

FILE NO.	FILE TYPE	PHYSICAL LENGTH	LOGICAL LENGTH	FIRST LINE	LAST LINE	WORDS IN COMMON	FILE DESCRIPTION
0	4	300	29	0	0	0	. . . . .
1	3	300	137	10	190	0	. . . . .
2	3	300	136	100	240	0	. . . . .
3	3	300	39	10	50	0	. . . . .
4	3	300	50	120	190	0	. . . . .

BKSPACE STATEMENT

This statement backspaces the cassette or floppy disk to the beginning of the preceding file header.

SYNTAX:

BKSPACE [#select code]

ok rpm 5/3/1981

EXAMPLE:

BKSPACE #N

where N is the select code of the cassette or floppy disk to be backspaced. If the cassette referenced is on clear leader at either the beginning or the end of tape, an ERROR 58 will result.

USAGE EXAMPLE:

BKSPACE is most convenient for returning the tape to the beginning of a file header after executing a FILEID.

```
10 DIM A$[7] ...
.
100 FILEID A$[1]
110 BKSPACE
.
```



```
110 SERROR 58:GOTO
```

FIXED AND FLOAT STATEMENTS

The FIXED and FLOAT statements provide the optional capability to define their parameters by expressions. Thus, the output format may be determined by a calculation or user input. The expression must evaluate to a number between 0 and 11. If the number is not an integer, it is rounded.

SYNTAX:

FIXED expression
FLOAT expression

USAGE EXAMPLES:

```
10 PRINT PI
20 FIXED 3.5
30 PRINT PI
40 K=5.3
50 FLOAT K
```

PRT-ALL STATEMENT [Cont.]

Notice that the program appears to run properly but, in fact, gives an incorrect answer for X=0. When discovered, it is assumed that there has been an error in the calculations; therefore the PRINT-ALL mode is used. This, however, results in all the ERROR 58's being displayed. The solution, then, is to put a PRT-ALL 1 in line 170 and a PRT-ALL 0 in line 215. This will cause the computer to print only those error messages which occur between lines 170 and 215. Once it has been determined that the problem is an ERROR 103 (division by zero), the statement "PRT-ALL X=0" may be inserted to further define the problem. This will cause the PRINT-ALL mode to be turned on only when X is zero.

```
RUN
ERROR 58 IN LINE 150
ERROR 58 IN LINE 150
ERROR 58 IN LINE 150
ERROR 58 IN LINE 150
ERROR 58 IN LINE 150

175 PRT-ALL 1
215 PRT-ALL 0
RUN
ERROR 58 IN LINE 150
ERROR 103 IN LINE 210
```

ADVANCE STATEMENT

The ADVANCE statement is a mass memory related statement which may be used to move the data pointer a specified number of items either forward or backward within a file. Because ADVANCE works on the pointer directly, mass memory operations are substantially accelerated.

SYNTAX:

ADVANCE # file number; number of items, return variable

ok Ben 5/3/1981

EXAMPLE:

ADVANCE #N;X,E

where N is the number of the file in the FILES statement, X is the number of items to be skipped, and E is a variable in which a test of completion is returned.

If X is positive, the pointer is moved X items forward in the file. If the value of X is negative, the pointer is moved back X items toward the beginning of the file. If X is zero, the statement has no effect. The value of X must be within the range $-32767 \leq X \leq 32767$.

If the value of E after executing ADVANCE is zero, the operation was successful. If either the beginning or an end of file was encountered, the return variable will contain a value equal to the number of items requested to be skipped, less the number actually skipped. Thus, if the data pointer is at the next to the last item in a file, and X is equal to 3, the variable E would contain the value 1. This is due to the fact that after the pointer was moved two items forward, one more item remained to be skipped. The return variable will be negative if the ADVANCE was in the reverse direction.

USAGE EXAMPLE:

The following program reads names and social security numbers from each record of a file. The name and social security number are, respectively, the fifty-first and sixty-second items in each record. Each record contains one hundred twenty items.

```
100 DIM N$[30],S$[9]
110 FILES PAYFIL
120 ADVANCE #1;50,X
130 READ #1;N$
140 ADVANCE #1;10,X
150 READ #1;S$
160 PRINT N$,TAB(35),S$
190 ADVANCE #1;108,K
210 IF  NOT K THEN 130
220 PRINT
230 PRINT
240 END
```

Line 210 tests for end of file. If the variable is zero, the ADVANCE has been completed and, therefore, there are more items in the file. If the variable is non-zero, the ADVANCE was not completed and the last item in the file has been reached.

Does not work BFM 5/31/1981

UPDATE STATEMENT

The UPDATE statement is used to update specific items within a record without the need to re-write the whole record. When used in combination with the ADVANCE statement, UPDATE provides a powerful means of manipulating individual items in mass memory files.

SYNTAX:

UPDATE #file number; item [,item, . . . ,item]

EXAMPLE:

UPDATE #N;A,B,C

where N is the file number of the mass memory file to be updated, and A, B, and C are data items to replace existing data. The type of each item in the list must be **EXACTLY** the same as that which was written by the original PRINT# statement. This means that numbers to be written must be of the same precision, and any string which is updated must be of the same length as the string which is updated. If these requirements are not met, ERROR 98 results.

USAGE EXAMPLES:

UPDATE may be used to change an item without having to first read that item. The following program segment is from a routine in a payroll system. It reads the name, social security number, number of hours worked, and pay rate from a file. It then calculates the gross pay and prints all information while it updates the payroll file. The program does not read gross pay; it merely updates it. Notice that gross pay is the only item written. It is not necessary to rewrite the name, social security number and pay rate.

```
10 DIM N$[30],S$[9]
110 FILES PAYFIL
120 IF END#1 THEN 180
130 READ #1;N$,S$,R,H
140 LET P=R*H
150 PRINT N$,TAB(35),S$,TAB(50),H,P
160 UPDATE #1;P
170 GOTO 130
180 PRINT "END OF FILE."
190 PRINT
200 END
```

The UPDATE statement in line 160 writes P (pay) after H (hours). Because UPDATE is used instead of PRINT#, only one item in the record is written.

FSIZE FUNCTION

The FSIZE function is used to determine the number of physical records in a mass memory file.

SYNTAX:

FSIZE(file number)

ok 3pm 5/13/1981

EXAMPLE:

K = FSIZE(F)

where F is the number of the file as previously specified in the FILES statement.

USAGE EXAMPLE:

```
10 FILES DATAF,*
20 DIM A$[6]
30 DISP "NAME OF BACKUP FILE";
40 INPUT A$
50 OPEN A$,FSIZE(1)
60 DISP "BACKUP FILE "A$"CREATED."
```

REC FUNCTION

REC returns the number of the record currently being accessed in the file number specified of the mass memory system.

SYNTAX:

REC(file number) 6k mem 5/31/91

EXAMPLE:

K = REC(F)

where F is the number of the previously specified mass memory file in the FILES statement.

USAGE EXAMPLES:

```
DISP REC(F)
```

RWORDS FUNCTION

The RWORDS function returns as its value the number of words remaining in the current record of the mass memory file which is specified as the argument of the function. The result will always be 256 or less.

SYNTAX:

RWORDS(file number) okBem 5/31/1981

EXAMPLE:

K = RWORDS(F)

where F is the number of a file as previously specified in a FILES statement.

USAGE EXAMPLE:

```
10 DIM AI[120]
20 FILES MYFILE
 .
170 K=RWORDS(1)/2
180 FOR I=1 TO K
190 PRINT #1;A(T)
200 T=T+1
210 NEXT I
 .
```

SUNIT FUNCTION

The SUNIT function may be used to determine the current mass memory unit number being referenced by the computer and to change the unit number without destroying existing file definitions.

SYNTAX:

ok checked Apr. 1980 3pm

SUNIT(unit number)

EXAMPLE:

K = SUNIT(U)

The current unit number is always returned as the value of the function. In addition, if the argument (U) is positive, the current unit number is changed to U. If U is negative, the current unit number remains the same. U must be an integer less than 4.

Because the SUNIT function does not destroy file definitions, data on mass memory files, on either platter, may be accessed without directory seeks. This can significantly accelerate disk operations.

USAGE EXAMPLES:

The following program copies a data file named "DF1" from unit 0 to unit 1.

```
100 UNIT 0
110 FILES DF1,*
120 LET K=SUNIT(1)
130 ASSIGN "DF1",2,C
140 LET I9=FSIZE(2)
150 FOR I=1 TO I9
160 K=SUNIT(0)
170 READ #1,I;A,B,C,D,E,F
180 K=SUNIT(1)
190 PRINT #2,I;A,B,C,D,E,F
200 NEXT I
210 DISP "DONE."
220 END
```

Line 100 sets up the first unit number (0). Line 110 defines the file number for the first unit number and a file position that will be used with the second unit (1). The system is switched to unit 1 by line 120. Line 130 defines file #2 on unit 1. Lines 160 and 170 read from unit 0, while lines 180 and 190 print the same information on unit 1.

Lines 160 and 180 change the unit numbers without affecting the files definition of lines 110 and 130.

SUNIT FUNCTION [Cont.]

In the preceding example, it is mandatory that file #1 is not referenced when set to unit 1 and file #2 is not referenced when set to unit 0. This rule cannot be violated even if the file names are identical.

In general, the unit number currently defined when executing a FILES or ASSIGN statement must be referenced by the SUNIT function prior to subsequent references to insure that unit numbers and files are properly associated.

CAUTION

Extreme caution must be exercised when the SUNIT function is used to change the referenced unit number. The system does not keep a record of the platter on which a particular file resides. It is up to the user to ensure that the correct unit number is specified for the file being referenced. Failure to do so may result in loss of data on one or both disks.

XREF# COMMAND

The XREF# provides the optional capability to output a cross-reference to a specified device select code. The XREF# command works in conjunction with either the Advanced Programming I or FAST BASIC II ROM. One of these ROMS must be installed to produce the cross-reference listing.

SYNTAX:

XREF# select code

6K ROM used from since ± April 1980

EXAMPLE:

XREF# 9

The select code may specify a secondary printer, a paper tape punch, or any other peripheral which can accept data. By reading a punched tape back into the 9830, and sorting or otherwise processing the cross-reference listing, an alphanumerically ordered list or any similar utility is practical.

USAGE EXAMPLE:

```
XREF #2
```

Note that the FAST BASIC I ROM must be in a higher priority location relative to API for the select code specification to function. The following table lists the priority order from highest to lowest.

PRIORITY ORDER		LOCATION
Highest	1	External First Slot (Top)
	2	External Second Slot
	3	Internal Third Slot (Front)
	4	Internal Second Slot (Middle)
	5	External Fifth Slot (Bottom)
	6	Internal First Slot (Rear)
	7	External Fourth Slot
Lowest	8	External Third Slot (Middle)

DCAT# COMMAND

The DCAT# command provides the optional capability to output a mass memory catalog to a specified select code. DCAT# can produce a catalog on removable media such as punched paper tape. The data can then be read back into the computer for sorting or other processing.

SYNTAX:

DCAT# select code

EXAMPLE:

DCAT# 9

Refer to the mass memory manual for further information related to the CATALOG command.

LISTALL COMMAND

The LISTALL command is used to de-secure a program in memory and to take the computer out of "secure mode." The command does not actually list the program, but it makes a secured program available for listing or editing. Because the secured mode is disabled, it is possible to store the de-secured program as a type 3 (not secured) file.

SYNTAX:

LISTALL

ok mem checked ± April 1980
also 5/31/1981

USAGE EXAMPLE:

```
LIST

10 PRINT "THIS IS A SHORT";
20 PRINT "PROGRAM."
30 END
SEC
LIST

10 *
20 *
30 *

LISTALL
LIST

10 PRINT "THIS IS A SHORT";
20 PRINT "PROGRAM."
30 END
```

APPENDIX A

ADVANCE

ADVANCE # file number; item skip count, return variable

Positions the data pointer the specified number of **ITEMS** preceding or following the current position.

BKSPACE

BKSPACE [# select code]

Positions the tape to the beginning of the preceding file header.

COL

COL(array name)

Returns the current working COLUMN dimension of the specified array.

COMPARE

COMPARE array name (subscripts) **TO** array name (subscripts), return variable

Compares the contents of one array to the contents of a second array beginning at a specified location in each array. The comparison continues to the end of the arrays and returns the number of unequal corresponding **WORDS.**

DCAT#

DCAT# select code

Provides the optional capability to output a mass memory catalog to a specified select code.

DEC

DEC(expression)

Converts a decimal integer to its octal equivalent.

EXOR

EXOR(expression 1 expression 2)

Provides the result of an "exclusive or" of two 16-bit integer values or expressions.

FILEID

FILEID [# select code,] array name (subscripts)

Reads the items contained in a file header into a specified array.

APPENDIX A [Cont.]

FIXED

FIXED expression

Permits an expression to define the number of decimal places in a fixed format.

FLOAT

FLOAT expression

Permits an expression to define the floating format.

FRAC

FRAC(expression)

Returns the fractional portion of an expression.

FSIZE

FSIZE(file number)

Returns the size of a data file on the 9880 mass memory system in records.

LISTALL

LISTALL

Permits listing of a secured program. Also permits storing the de-secured program as a non-secured program file.

MOD

MOD(expression 1, expression 2)

Returns the modulo of expression 1 and expression 2.

OCT

OCT(expression)

Converts an octal integer to its decimal equivalent.

PRT-ALL

PRT-ALL expression

PRT-ALL is a program statement that functions like the PRT-ALL button. When the expression evaluates to a 1, PRT-ALL is on. When it evaluates to **0**, PRT-ALL is off.

REC

REC(file number)

Returns the record number currently pointed to by the 9880 mass memory system.

ROW

ROW(array name)

Returns the current working ROW dimension of the specified array.

RWORDS

RWORDS(file number)

Returns the number of remaining words in the record currently pointed to by the 9880 mass memory system.

SCAN

SCAN array name $\left\{ \begin{matrix} > \\ < \\ = \\ \neq \end{matrix} \right\}$ match expression, return variable [, mask expression]

Scans an array for an element which satisfies any of four possible relational operators against a match expression, and returns a pointer to the element which satisfies the relation. A mask may be applied against each element of the array being scanned.

SEND

SEND array name (subscripts) **TO** array name (subscripts)

Moves the contents of a source array beginning at a specified location into a destination array beginning at a specified location.

SUNIT

SUNIT(unit number)

Returns the current unit number of the 9880 mass memory system, if the parameter is negative. Additionally, this function will change the currently addressed unit number **WITHOUT** destroying existing file definitions, if the parameter is positive.

SWAP

SWAP $\left\{ \begin{matrix} \text{string name} \\ \text{variable name} \\ \text{array name} \end{matrix} \right\}$, $\left\{ \begin{matrix} \text{string name} \\ \text{variable name} \\ \text{array name} \end{matrix} \right\}$

Swaps the entire contents of any two arrays, strings or simple variables specified.

TRIG

TRIG(dummy argument)

Returns a number defining the current trigonometric mode of the machine. The return is 0 for degrees, 1 for radians, and 2 for grads.

APPENDIX A [Cont.]

UND

UND(variable or array element)

UND returns a 1 if the named variable is undefined. It returns a 0 if it is defined.

UPDATE

UPDATE# file number; item, [item, . . . , item]

Permits modification of a data item without rewriting the entire record of a 9880 mass memory file.

XREF#

XREF# select code

Provides capability for device select code specification.

APPENDIX B
ARRAY INITIALIZATION AND STRUCTURE

When using the SEND, SCAN, or COMPARE statements, it is useful to be aware of the way the 9830 stores arrays. Arrays are stored on a row-by-row basis in memory. A subscripted reference to an array variable may be interpreted as A(R,C) where R is the row number and C is the column number. For an array A(3,3), the storage order is:

A(1,1) A(1,2) A(1,3) A(2,1) A(2,2) A(2,3) A(3,1) A(3,2) A(3,3)

For each element of A, there may be one, two, or four words allocated, depending on whether A is an integer, split, or full-precision array. This fact must be taken into account when using array operations of FAST BASIC I

SEND

The operation of SEND is described in the flow chart, Figure B1. The detailed mechanics of the element manipulations are shown in Figure B2.

1. Compute number of words from A to B.
2. Compute number of words from C to D.
3. Determine the smallest range from steps 1 and 2 above, M words to be moved.
4. Test for integral of four words.
5. Move M words from A1 through A1+M to C1 through C1+M.

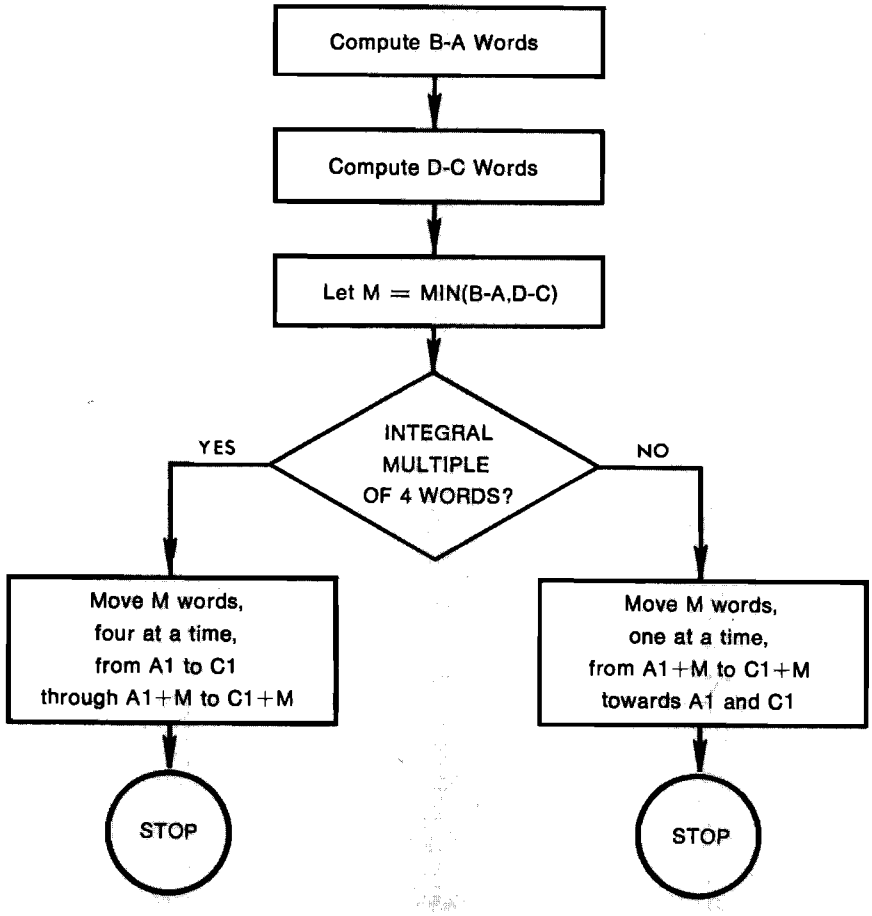


Figure B1. Flowchart of SEND

APPENDIX B [Cont.]
ARRAY INITIALIZATION AND STRUCTURE

When initializing large arrays, the SEND statement is much faster than the MAT = CON or MAT = ZER statements. Because SEND has two modes of operation, attention must be given to methods for pre-initialization.

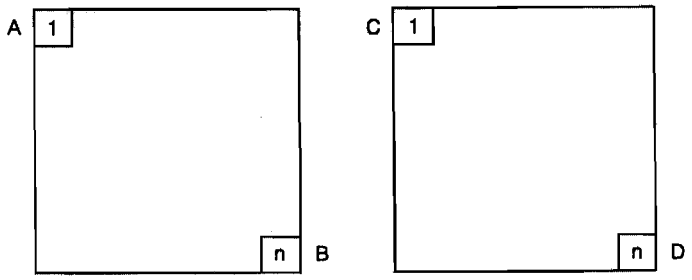
When the number of **words** in an array to be initialized is an integral multiple of four, SEND moves four words at a time. Accordingly, the first four words of the array must be pre-initialized with the value to which the rest of the array is to be initialized. Split-precision arrays require two, and full-precision arrays need only one, **element** pre-initialized. The destination in the SEND statement used to initialize the array must specify the fifth word of the array. For integer-precision arrays, A(5) or A(1,5) is correct. For split-precision arrays, A(3) or A(1,3) is correct, and for full-precision arrays, A(2) or A(1,2) are the correct destination subscripts.

It is advantageous and faster to dimension arrays in an integral multiple of four words whenever SEND is to be used, either for data transfer or initialization.

When the array size is not an integral multiple of four words, SEND moves one word at a time, starting from the last element of the array. The words to be moved are transferred from the bottom to the top of the array. The last element of the array being moved is the one to be pre-initialized. Note that if the destination array is smaller than the source array, the element to be pre-initialized will not be the last physical element of the source array, but rather the element corresponding to the last element in the destination array.

COMPARE

As the COMPARE statement compares arrays on a word-by-word basis, the number that the statement returns will be the number of **words** that are not equal. For split or full-precision arrays, each pair of unequal elements may have more than one word which is not equal. The only returns that are meaningful for non-integer arrays are zero and non-zero. A zero indicates that the arrays are equal, and non-zero means that the arrays are different by at least one element. If a number greater than one is returned, then the arrays are unequal by an unknown number of ELEMENTS.



Source Array **Destination Array**

where,

A = First **Referenced** Word of the Source Array
B = Last Physical Word of the Source Array
C = First **Referenced** Word of the Destination Array
D = Last Physical Word of the Destination Array

Figure B2

GLOSSARY

Argument — the number or value on which a function operates. 3.14 is the argument in "SIN(3.14)."

Directory Seek — in mass memory operations, the action of the disk drive in reading the file directory to determine the location of a particular file.

Element — a subscripted reference to an array. A(3,5) is an element of array A.

File Header — an area of tape preceding the data portion of every file on cassette or floppy disk. The header contains data descriptive of the file.

Mask — an expression which is converted to binary and logically AND'ed with an array element.

Match Expression — the expression which specifies the desired target.

Pointer — points to the next item to be accessed from a disk data file.

Precision — full-precision 12-digit accuracy uses four words of memory. Split-precision 6-digit accuracy uses two words of memory. Integer-precision limited to the range of −32767 to +32767 and uses one 16-bit word of memory.

Return Variable — a variable which contains the value computed as a result of an operation.

Sequential Location — the location of an array element relative to the first location in the array.

Word — the basic unit of information used by the 9830 to organize the read-write memory into 16-bit units.

NOTES

INFOTEK FAST BASIC I

INSTRUCTION MANUAL

Infotek Systems

1400 N. Baxter St. • Anaheim, Calif. 92806 • (714) 956-9300 • TWX 910-591-2711

5007703 PRINTED IN U.S.A.