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

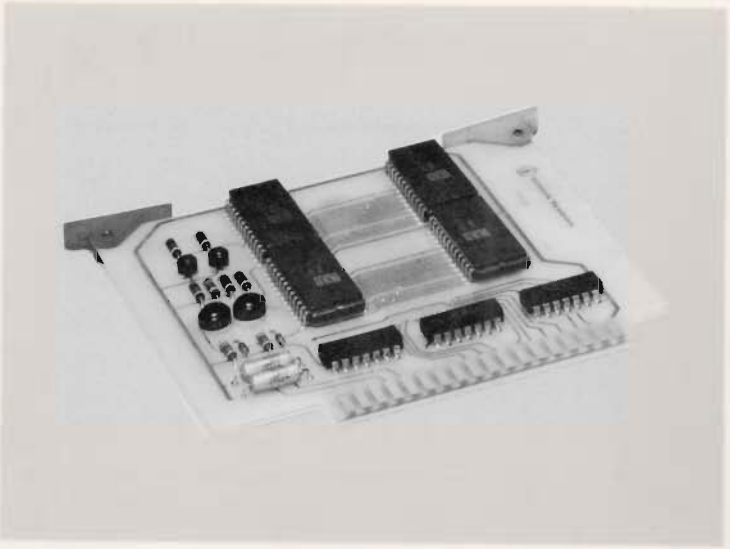
INSTRUCTION MANUAL

Infotek Systems

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

INFOTEK FAST BASIC III ROM

FAST BASIC III ROM
(EXTERNAL)



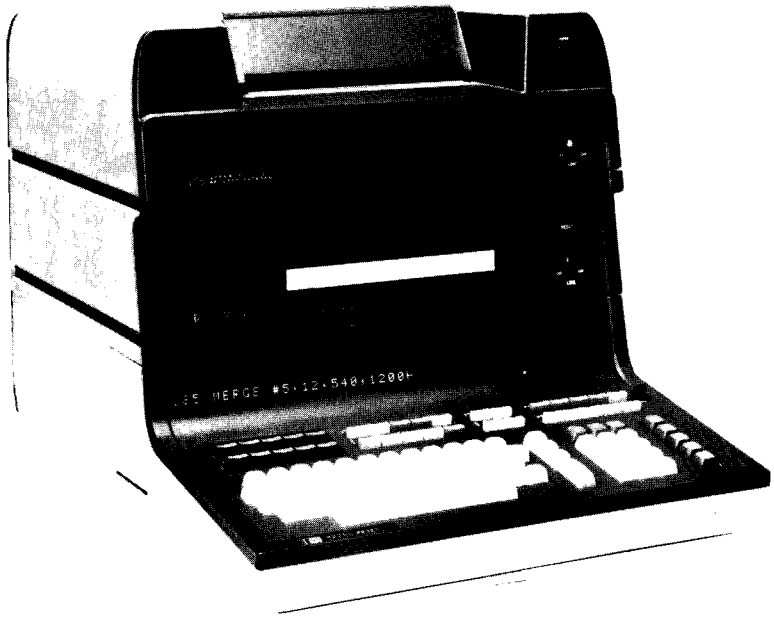
FAST BASIC III ROM
CIRCUIT CARD
(INTERNAL)

Johan E. Verbaarschot
1043 LYTTON ST.
N. VANCOUVER, B.C. V7H 2A7

INSTRUCTION MANUAL

FAST BASIC III

INFOTEK FAST BASIC III 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

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

INTRODUCTION

The Fast Basic III ROM is the most powerful of Infotek's Fast Basic series.

Fast Basic III allows an array to be redimensioned larger than 256 elements on a side. This substantially broadens the practical applications of the 9830 as it is now possible to sort, search, and otherwise reference an array of any size that will fit in memory.

Sorting and searching large data bases previously required tedious and lengthy programming. Now these tasks can be performed at speeds rivaling new machines and with only one or two lines of program.

Other important new capabilities for your 9830 with Fast Basic III include: pseudo-live keyboard, suspend a running program for later completion, and a means for a peripheral to interrupt the 9830 to demand service. The live keyboard permits interrogation and alteration of variables as well as transfer of control without stopping the program. Further, almost all keys can be used to provide over 200 distinguishable conditions.

The memory dump feature suspends a running program to tape or floppy. The suspended program can be recalled on the same or different 9830 for later completion. Program, all pointers, variables, and even 9880B Mass Memory operations are saved.

A peripheral such as a clock or other digital output instrument can interrupt the 9830 and issue instructions via the I/O bus, which the 9830 interprets as keystrokes. Upon interrupt the machine executes statements, functions, and commands as though manually entered via the keyboard.

SYNTAX CONVENTIONS

Brackets [ ] —items enclosed in brackets are optional.
Color —all other items must appear as shown.
Braces { } —one item enclosed in braces must be selected.

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 ROMs. 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.

NOTE: The Fast Basic III ROM allocates 6 words of read-write memory. The users available memory will be reduced by 6 words.

CAUTION

When power is first applied to the computer after installation of the FAST BASIC III 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 for your Infotek representative.

INSTALLATION [Cont.]

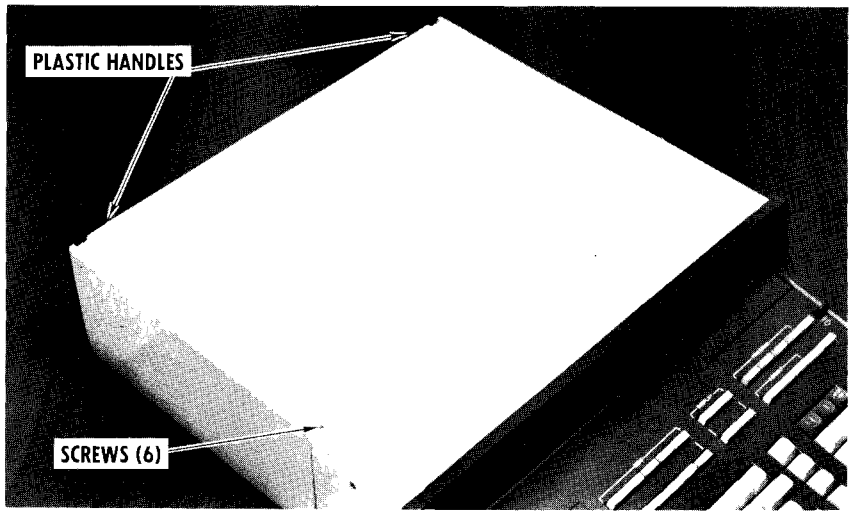


Figure 1.
Removal of Cover

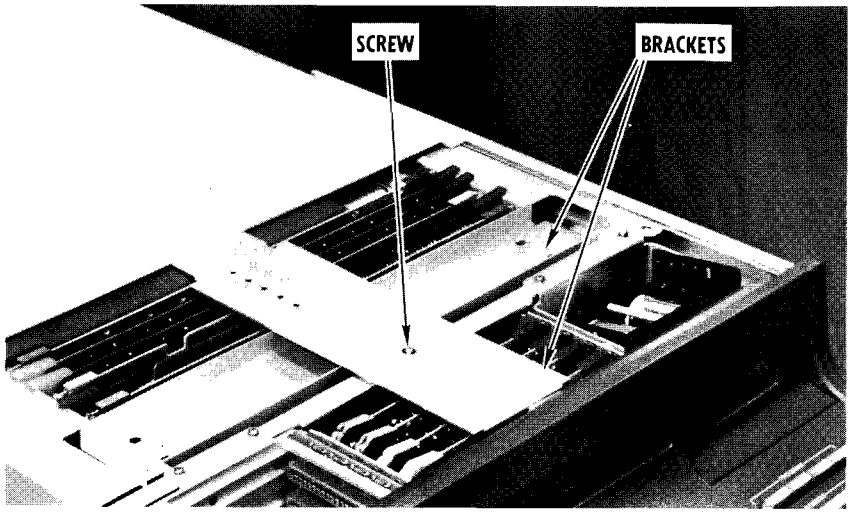


Figure 2.
Removal of Brackets

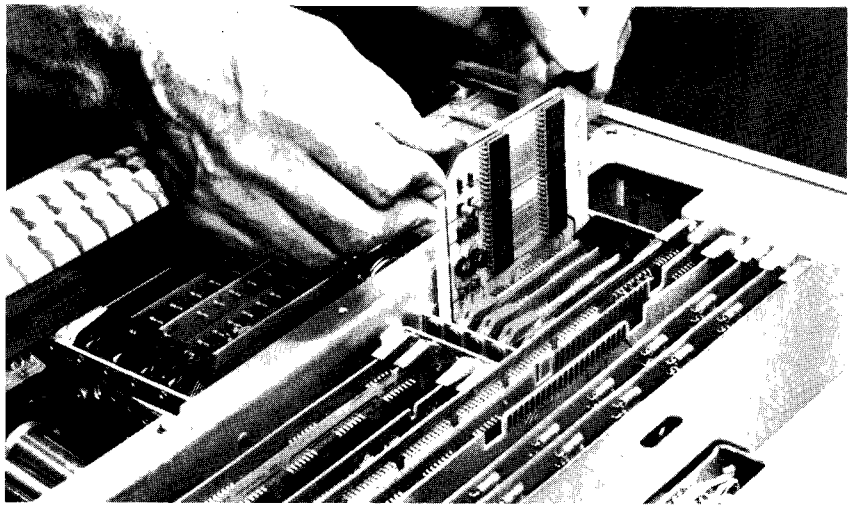


Figure 3.
Location of
FAST BASIC III ROM
Circuit Card

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

VDIM STATEMENT

With this statement, the limitation of 256 on rows or columns in the 9830 BASIC can be circumvented. VDIM redimensions an array so that its VIRTUAL (or apparent) dimensions may be any number of rows or columns which do not exceed the number of elements in the original array.

VDIM redimensions an array to a VIRTUAL size in preparation for the special VSORT, VSEARCH, and other operations carrying the V (VIRTUAL) prefix in the Fast Basic III ROM. A VDIM is also required prior to using the ARRAY statement. VDIM requires the Matrix ROM.

SYNTAX:

VDIM array name (rows [, columns])
VDIM is in effect only for the array specified in the most recently executed VDIM statement.*

The total number of elements in the VDIM statement must not be greater than the physical size of the array including the row-column product. Only one VDIM can be in effect at any one time. Executing a new VDIM statement replaces the previous statement. The array can always be referenced normally, regardless of VDIM execution.

The function of VDIM is similar to the REDIM statement in the Matrix ROM. Error 67 will result if the number of elements in the VDIM statement exceeds the physical size of the specified array.

*When more than one array has the same number of elements, the same number of dimensions, and the same precision, an explicit VDIM for an array is also an implicit VDIM for all other arrays meeting the same criteria.

USAGE EXAMPLE:

```
10 DIM A[250,4]
 .
 .
100 VDIM A[1000,1]
```

Line 10 dimensions array A to be 250 rows by 4 columns. Line 100 redimensions array A to have a VIRTUAL dimension of 1,000 rows by 1 column. Sorts and Searches of the VDIM array are now possible by using the VSORT and VSEARCH procedures described in the following paragraphs.

VSORT STATEMENT

When used in conjunction with VDIM, a sort can be performed on any array using VIRTUAL dimensions. Sorts can be performed on any row or column while maintaining array row/column integrity. VSORT requires the Matrix and APII ROMS. Refer to Chapter 3 of the APII manual for details relating to sort.

SYNTAX:

VSORT array name, {R or C} , # [,# [,# [,# [,# [,#]]]]]
Array name — must be an array that was viritually dimensioned by a VDIM statement.
R/C — Row or Column
#—The row or column number that will be sorted. A maximum of 6 rows or columns may be specified in a VSORT statement. The array will be sorted in the same sequence as the specifications in the statement. All conventions of sorting set forth for the APII ROM apply to VSORT.

SYNTAX EXAMPLE:

```
VSORT A,C,2
```

Array A will be sorted by column 2.

USAGE EXAMPLE:

```
10 DIM A[250,4]
20 LOAD  DATA #5,1,A
30 MAT PRINT A;
40 VDIM A[500,2]
50 VSORT A,C,2
60 MAT PRINT A;
70 END
```

Line 40 converts this array to 500 rows by two columns. Line 50 sorts array A by column 2. Lines 30 and 60 print the matrix before and after the VSORT.

VSORT STATEMENT [Cont.]

USAGE EXAMPLE:

Assume that it is necessary to contiguously sort the contents of three 1,000-word arrays. This program requires the Matrix, Advanced Programming II, and Fast Basic I ROMs.

```
10 DIM A[4,250],B[12,250],E[10,4]
20 LOAD  DATA #5,A
30 SEND A[1,1] TO B[1,1]
40 LOAD  DATA #5,2,A
50 SEND A[1,1] TO B[5,1]
60 LOAD  DATA #5,3,A
70 SEND A[1,1] TO B[9,1]
80 VDIM B[750,4]
90 SEND B[1,1] TO E[1,1]
100 MAT PRINT E
110 VSORT B,C,1
120 VSORT B,C,1,2,3,4
130 SEND B[1,1] TO E[1,1]
140 MAT PRINT E
150 END
```

Line 10 dimensions a working array (A) to be 4 by 250, a storage array (B) to be 12 by 250, and a printer array (E) to be 10 by 4. Since three data files are to be sorted, B is three times larger than A. Lines 20, 40, and 60 load data from select code five, files 1, 2, and 3, respectively. Lines 30, 50, and 70 SEND data from array A to the appropriate location in array B. When array B has been filled with the data, line 80 virtually dimensions array B to 750 rows by 4 columns. As VDIM functions like REDIM, array contents are rearranged in row-column order. Line 110 performs the primary SORT of the VIRTUAL array by column 1. Line 120 does the final sort. The APII manual explains in detail why two SORT statements execute faster than one.

Line 100 prints out the first 10 rows of unsorted data from array B. Line 130 moves the first 10 rows of sorted data from the VIRTUAL array B to array E. Line 140 prints the first 10 rows of the sorted data.

VSEARCH STATEMENT

The VSEARCH statement provides a fast and easy way to search specified rows or columns of a VIRTUAL array for specific values. VSEARCH requires the Matrix and APII ROMs.

SYNTAX:

VSEARCH array name, {R or C}, #, match expression, return variable.
Array name — Array which was specified in a VDIM statement.
R or C — Row or column to be searched.
— Number of the row or column to be searched.
Expression — Value to be searched for.
Return Variable — Returns the FIRST location in the specified row or column where the value is found. If the value is not found, the return variable is zero.

Does not work. Display blanked & keyboard control lost when entered. rfc 12/3/1981

USAGE EXAMPLE:

```
VSEARCH A,C,2,152,X
```

Line 40 will search column 2 of the VIRTUAL array for the value 152 and, if found, return to the location of first occurrence in X.

USAGE EXAMPLE:

```
10 DIM A[250,4]
20 LOAD  DATA #5,1,A
30 VDIM A[1000,1]
40 DISP "SEARCH TARGET VALUE";
50 INPUT V
60 VSEARCH A,C,1,V,X
70 IF X#0 THEN 110
80 PRINT V"NOT FOUND"
90 BEEP
100 GOTO 40
110 PRINT V"IS FOUND IN ELEMENT"X
120 GOTO 40
130 END
```

NOTE: Assume array A is zero except for the following:
A(4,4)=1
A(87,3)=2
A(143,3)=9

VSEARCH STATEMENT [Cont.]

When the program is run and the values 1, 2, 5, and 9, respectively, are input, the following printout results:

```
1     IS FOUND IN ELEMENT 16
2     IS FOUND IN ELEMENT 347
5     NOT FOUND
9     IS FOUND IN ELEMENT 571
```

Line 60 searches for a value (V) and returns its location in X.

USAGE EXAMPLE:

A company has 1,000 parts each with four items of information. The part number is in column 1 and the quantity on hand is in column 2. It is desired to determine the quantity on hand of a given part number.

```
10 DIM A[200,4],B[200,20]
20 FOR N=1 TO 5
30 LOAD  DATA #5,N,A
40 SEND A[1,1] TO B[(N-1)*40+1,1]
50 NEXT N
60 VDIM B[1000,4]
70 DISP "ENTER PART NUMBER";
80 INPUT P
90 VSEARCH B,C,1,P,X
100 IF  NOT X THEN 140
110 ARRAY B[X,2],Q
120 PRINT "PART NUMBER"P","Q"ON HAND
130 GOTO 70
140 PRINT "PART NO"P"NOT ON HAND"
150 BEEP
160 GOTO 70
170 END
```

Line 10 dimensions array A containing 200 items and array B capable of containing 1,000 items. Lines 20 to 50 set up a for-next loop which consecutively loads data files 1 thru 5 into array A. Array A is then sent to the appropriate location of array B such that the five consecutive 200 item files are loaded into the single 1,000 item file array B. Line 40 performs the SEND operation of array A to array B. Note the algorithm for incrementing the row number of array B for each consecutive array A file. Line 60 virtually dimensions array B to be 1,000 by 4. Lines 70 and 80 initialize the part number, variable P. Line 90, in accordance with the SEARCH conventions of APII, does a VIRTUAL search of array B by column 1 for the value of P and returns its row number in X. Line 100 branches execution to line 140 if the part is not found. The ARRAY statement of Fast Basic III returns the value of column 2, row X in the return variable Q, the quantity on hand. (See page 11 relative to the ARRAY statement.) Line 120 prints the information and 130 returns back to the input line 70. Line 140 is the "part not found" message followed by a beep and then branches back to another part number input.

ARRAY STATEMENT

The ARRAY statement returns the value of a specified element in a virtually dimensioned array. Optionally, ARRAY may be used to alter the value of the specified element. This statement functions only when the named array has been virtually redimensioned. ARRAY requires the Matrix ROM.

SYNTAX:

ARRAY array name (subscripts), return variable [, expression]
Array name — must be an array currently specified by VDIM explicitly or implicitly.
Subscripts — must be within the limits of the VDIM dimensions. The ARRAY statement must specify the same number of subscripts as contained in the original array dimension. The subscripts may be expressions.
Return Variable — A simple or subscripted variable.
Optional Expression — May be any expression and will supplant the value of the specified element. When the optional expression is used the return variable will still return the initial value of the element.

SYNTAX EXAMPLE:

```
ARRAY A[J,K],X,Y
```

The J row, K column value of array A will be returned in X and supplanted by Y.

USAGE EXAMPLE:

```
10 DIM A[250,4]
20 LOAD  DATA #5,1,A
30 VDIM A[1000,1]
40 DISP "ELEMENT NUMBER";
50 INPUT Y
60 ARRAY A[Y,1],X
70 PRINT X
80 GOTO 40
90 END
```

Line 30 virtually dimensions the array to 1,000 rows by 1 column. Line 60 returns in the variable X the element specified by Y.

ARRAY STATEMENT [Cont.]

USAGE EXAMPLE:

```
10 DIM A[250,4]
20 LOAD  DATA #5,1,A
30 VDIM A[1000,1]
40 DISP "ELEMENT NUMBER TO UPDATE";
50 INPUT Y
60 DISP "NEW VALUE";
70 INPUT X
80 ARRAY A[Y,1],Z,X
90 PRINT "ELEMENT NUMBER"Y"OLD VALUE"Z"UPDATED VALUE"X
100 GOTO 40
110 END
```

Lines 10, 20, and 30 initialize the array, load data in the array, and VDIM the array to 1,000 rows. Lines 40 and 50 specify the element number to update and lines 60 and 70 specify the new value, X. Line 80 updates the value in row Y, column 1 to the new value, X. Line 90 prints a record of the original and new value and line 100 branches for another update.

USAGE EXAMPLE:

Array T contains year-to-date payroll data for 800 people. Array A contains the current period data. It is desired to add only the gross pay column (3) of array A to array T. Because the first column of the array contains the employee number and the 4th column contains the department number, Matrix addition cannot be used.

```
10 DIM A[200,16], T[200,16]
 .
 .
100 VDIM A[800,4]
110 FOR N=1 TO 800
120 ARRAY A[N,3],X
130 ARRAY T[N,3],Y,X+Y
140 NEXT N
 .
 .
500 END
```

VROW AND VCOL FUNCTIONS

The two functions called VROW and VCOL take an array name as an argument and return the VIRTUAL dimensions of the array. These functions are similar to the ROW and COL functions of Fast Basic I, which return the normal working dimensions of arrays regardless of whether they have been referenced by a VDIM statement.

SYNTAX:

VROW (array name)
VCOL (array name)

VPRINT AND VREAD STATEMENTS

The VPRINT and VREAD statements are incorporated for operation with an HP 9880B Mass Memory system using the Infotek Mass II ROM. The statements operate similarly to the MAT PRINT and MAT READ instructions. The advantage lies in the ability of the VPRINT and VREAD to work with the VIRTUAL dimensions of an array so that arrays of any arbitrary number of elements can be processed.

SYNTAX:

VPRINT# file no. [,record no.] ; array name
VREAD # file no. [,record no.] ; array name

The operation of VPRINT and VREAD is identical to MAT PRINT and MAT READ as described in the HP 9880B Mass Memory manual. The only exception is that only one array name may be referenced by a VPRINT or VREAD statement.

If the Infotek Mass Memory II ROM is not used, Error 94 will result.

MREAD STATEMENT

MREAD allows the 9830 to read data at rates up to 10,000 bytes per second. Packing density per word, the number of bytes to be read, and the destination array are also specified. MREAD requires the Infotek Fast Basic I ROM and the FI-30 High Speed TTL I/O Interface.

SYNTAX:

MREAD (select code, {1 or 2} ,byte count expression) array (subscripts)
SC — Any select code between 1 and 9 must be specified.
Packing — Specifies the packing density:
1 — Specifies 1 byte per word.
2 — Specifies 2 bytes per word.
Count — A positive number specifies the number of bytes to be read.
A negative number will terminate the read operation upon receipt of a line-feed character. The line feed character will not be stored in the array.
The value of the byte count is ignored.
Array () — Specifies the destination array. The beginning subscripts must be specified. The subscripts may be expressions.

SYNTAX EXAMPLE:

```
MREAD( 4,1,1000)A[1,1]
```

Data is read from device code 4 packing 1 byte per word. 1,000 bytes will be stored in array A beginning at row 1, column 1. As 1,000 bytes are to be read and packing is 1 byte per word, the array must be dimensioned to be at least 1,000 words in length. When packing 2 bytes per word, the array size in words may be one-half the specified number of bytes. Error 205 will result if the byte count exceeds the working size of the specified array.

USAGE EXAMPLE:

A high speed digital instrument outputs a measurement in a 6 character field consisting of a sign character, 4 numeric characters, and a carriage return-line feed. It is desired to take 1,000 readings as quickly as possible and print the data later. This can be easily acccomplished as shown in the following listing.

```
10 DIM A[100,30],A$[60]
20 MREAD( 6,2,6000)A[1,1]
30 FOR N=1 TO 100
40 TRANSFER A[N,1] TO A$
50 FOR P=1 TO 60 STEP 6
60 WRITE (15,100)VAL(A$[P,P+5]);
70 NEXT P
80 PRINT
90 NEXT N
100 FORMAT 10F8.0
110 END
```

MREAD STATEMENT [Cont.]

Line 10 dimensions array A to 3,000 words or 6,000 bytes. This is sufficient for 1,000 six character readings. A$ is dimensioned to be 60 characters or the equivalent of 10 consecutive readings. Line 20 performs the high speed read of 6,000 bytes from select code 6 (the instrument) packed at 2 bytes per word. As the bytes enter the 9830 they are packed in array A beginning at row 1, column 1, the first element. Lines 30 thru 100, a 100 iteration for-next loop, transfers the contents of array A row by row into A$ at line 50. Then with the 10 iteration for-next loop, beginning at line 60, the contents of A$ (in groups of six characters according to format line 110) are printed 10 readings across the page on 100 lines, a total of 1,000 readings.

If the readings were printed on a real-time basis, the reading of the instrument would be "printer bound". By bringing the readings into array A first, the entire operation of 1,000 readings can be accomplished in half of a second. Printing, of course, will take substantially longer.

Use of the FI-30 with the Fast Basic III ROM provides an approximate factor of 30 in speed increase over the HP 11202A working conjunction with a basic program.

MWRITE STATEMENT

MWRITE allows the 9830 to output data up to 10,000 bytes per second. Packing density, the number of bytes to output, and the source array are also specified. MWRITE requires the Infotek Fast Basic I ROM and the FI-30 High Speed TTL I/O Interface.

SYNTAX:

MWRITE (select code, { 1 or 2 } , byte count expression) array (subscriptions)
SC — Any device code between 1 and 9 is allowed.
Unpacking — Specifies the unpacking density:
1 — 1 byte per word from the lower 8 bits.
2 — 2 bytes per word.
Count — When positive — outputs the specified number of bytes.
When negative — outputs the specified number of bytes and adds the carriage return-line feed character when completed.

SYNTAX EXAMPLE:

```
MWRITE( 3,1,400)A[1,1]
```

Outputs to select code 3 and unpacks data at 1 byte per word. 400 bytes will be output from array A beginning with row 1, column 1.

As 400 bytes are to be output at 1 byte per word, the array must be 400 words or greater in length. When unpacking arrays at 2 bytes per word, the array must be dimensioned not less than one-half the byte count. Error 205 will result if the byte count exceeds the working size of the specified array.

USAGE EXAMPLE:

It is desired to diagnose a suspect logic analyzer in order to assess its accuracy. As the analyzer is suspected of making ocassional errors, a large volume of data must be processed in order to perform a valid test. Under these circumstances, the higher the speed (within the limits of the analyzer), the less time will be required to make an accurate determination of the analyzer accuracy.

The following program sends a random number array into the analyzer and then subsequently reads the data back from the analyzer, comparing the returned data to that written to determine the number of non-equalities between the two arrays.

```
10 DIM A[128,8],B[128,8]
20 FOR N=1 TO 128
30 FOR M=1 TO 8
40 A[N,M]=RND(1)*3E+04
50 NEXT M
60 NEXT N
70 MWRITE( 4,2,2048)A[1,1]
80 MREAD( 4,2,2048)B[1,1]
90 COMPARE A[1,1] TO B[1,1],X
100 IF  NOT X THEN 70
110 DISP "TEST FAILED,"X"ERRORS"
120 END
```

MWRITE STATEMENT [Cont.]

Line 10 dimensions two arrays of 1024 words each. Lines 20 thru 60 are a for-next loop which loads array A with 1024 random numbers between 1 and 30,000. Line 70 outputs the contents of array A beginning with the first element, unpacking at 2 bytes per word via select code 4 to the logic analyzer. The 2048 bytes will be output in approximately $^{2}/_{10}$ of a second. Following the output, line 80 brings the data back from the logic analyzer packing at two bytes per word in array B beginning in the first element. Line 90 compares array A to array B and returns the number of non-equalities, if any, in X. Line 100 tests X which if Ø, branches execution to line 70, and repeats the READ and WRITE statements.

The routine, if not stopped by an error, will execute about 100 times per minute. At these speeds, 2.4 megabytes of data will be verified every minute.

MDUMP STATEMENT

The MDUMP statement can be executed either from the keyboard or by a program. It enables a user to store the entire contents of memory including program, variables, data pointers, and flags. The data representing the suspended program can be loaded and execution continued as if the program had never been interrupted. MDUMP will work with the HP 9830 cassette, 9865A peripheral cassette, or any of the Infotek FD-30 Series Floppy Disk Systems.

SYNTAX:

MDUMP enable/disable mode expression

Expression — Any positive number causes immediate execution of MDUMP. However, in order for an MDUMP to occur, the number must coincide either with the internal cassette select code 10 or a peripheral cassette or floppy select code, 1 thru 9. If any number other than 1 thru 10 is used, the MDUMP statement is ignored and no diagnostic is issued.

If negative, the LOAD and STORE keys of the 9830 are temporarily redefined. If either key is pressed, the 9830 will execute the corresponding operation to the select code specified by a subsequently struck numeral key, Ø thru 9. For keyboard operation of MDUMP the internal cassette drive is referenced by the Ø key. Select codes 1 thru 9 directly correspond to keys 1 thru 9.

If Ø, the MDUMP mode of the machine is negated. All operations will be as though no MDUMP statement with a positive or negative argument had been executed. The LOAD and STORE keys resume their normal functions.

When MDUMP is followed by any positive expression in a program statement, the program will immediately suspend to the device specified by the expression.

In keyboard operation, the program may be either suspended or loaded. In order to enter MDUMP mode, it is necessary that an MDUMP −1 statement be excecuted. Subsequently, the program may be suspended by pressing the STORE button followed by a numeral key corresponding to the peripheral device to which the program is to be suspended.

To resume a suspended program, execute MDUMP — 1 statement from the keyboard, then follow by pressing the LOAD button, and then the numeral key corresponding to the device from which the suspended program is to be read.

If the specified peripheral is non-existent or not ready, the MDUMP statement will be ignored. If the specified peripheral is the wrong type, (such as a printer) the results will be unpredictable.

Using MDUMP to suspend a program requires that the dynamic remaining memory be not less than 512 words for systems using the HP 9880B Mass Memory. For all other configurations, only 256 words are required. If sufficient memory for program suspension is not available, the instruction will be ignored and no error message will be issued.

Diagnostic messages are not given as this would halt a program for reasons unrelated to the program itself. Execution of MDUMP can be determined by observing the peripheral. When executing, cassette motion can be seen and an FD-30 Series Floppy will illuminate the BUSY lamp.

except that D... condition is not

Some ... with STOP

MDUMP STATEMENT [Cont.]

A cassette tape intended for an MDUMP must be rewound and a floppy disc reset. Either media must have at least one file of any length marked, as MDUMP always operates in file Ø. MDUMP has been assigned file type 7 so that it may be distinguished in a TLIST of the media. If additional information is to be stored on the same media, then file Ø should be marked to a length equal to the nearest power of 2 of the machine memory in words. For example, an 8K machine which has a nominal working size of 7904 words should have a file of 8192 words marked. Using this formula, MDUMP will not overrun the file 1 header. A 9830 with Infotek's 16K EM-30 extended memory will require a file marked 16,383 words to ensure that the file 1 preamble is not overrun.

With the HP cassette system and the Infotek FD-30 Series Floppy Discs, on completion of an MDUMP, the tape is rewound or a floppy unit is reset. The MDUMP operation in STORE mode is followed by a power-up initialization routine and, therefore, if Infotek EM-30 memory and Fast Basic I and II are installed, the machine will execute the power-up routine.

When using MDUMP to load a suspended program, the HP 9830 will load the program and resume execution of the program and concurrently initiate a rewind of the cassette or floppy.

USAGE EXAMPLE:

```
100 MDUMP 5
```

Causes immediate suspension of the program to select code 5.

USAGE EXAMPLE:

```
MDUMP -1
```

Causes the machine to continue program execution, but listens to the STORE and LOAD keys which have now been redefined so that if pressed, the machine continues to listen for a numeral key. If the key next pressed is a numeral, Ø thru 9, the program will either STORE or LOAD depending on which of the two keys was depressed. Any other key will cancel the LOAD or STORE key prefix setting.

MDUMP was designed to provide the 9830 user with three important new capabilities. First, in conjunction with KEY (see page 20), it is possible to interrogate the degree of program completion without halting the program. Thus, if it becomes apparent that the completion of program execution will take more time than one has available, he has the option of pressing the STORE and device select code numeral keys, thereby suspending operation for later completion. The machine may then be put to a different task or turned off. Second, in conjunction with the SERVICE statement (see page 19), the MDUMP can be initiated via a specified interrupt interface in the event that prime power fails for such period of time that the batteries of the uninterruptable power supply are not capable of maintaining system operation. The Infotek UP-30 Series Uninterruptable Power Supplies all interface to the 9830 and automatically initiate MDUMP when the charge stored in the batteries is depleted to a remainder of 3 minutes.

Finally, the third application of MDUMP is crucial to the operation of the Infotek RT-30 clock which is capable of interrupting the 9830 at a specified time in order to perform any instructed task. If a program is running at the interrupt time, the clock can initiate an MDUMP so that the interrupted program can be completed at a later time automatically.

SERVICE STATEMENT

The SERVICE statement provides a means for a specified peripheral to interrupt the 9830. Upon interrupt, the 9830 interprets bytes output by the specified peripheral as key strokes. Therefore statements, functions, and even commands allowed from the keyboard only will be executed by the 9830 as though typed by an operator. SERVICE is particularly useful in conjunction with the Infotek RT-30 clock which can store up to 32 keystrokes of instructions to issue at the preset interrupt time. Use of the SERVICE statement requires that the specified select code peripheral interface be capable of activating the 9830 I/O bus interrupt line in addition to normal data transfer. The Infotek EI-30 External Interrupt Interface has this capability. Instruments interfaced via the EI-30 have the ability to demand service.

SYNTAX:

SERVICE Select Code

The select code may be an expression. SERVICE is in effect only for all declared select codes. The select code must be within the range 1 thru 9. SERVICE requires the Infotek EI-30 Interface for general applications. The Infotek RT-30 real-time clock works directly with the Fast Basic III ROM and does not require the EI-30. When SERVICE is executed, the 9830 interrupt table is flagged to interpret the specified select code data as keystrokes.

SYNTAX EXAMPLE:

```
SERVICE 9
```

If the interrupt line is activated, the peripheral device interface, select code 9, will have its output data interpreted as keystrokes.

USAGE EXAMPLE:

It is desired to load and run a program, file 12 at 3:30 p.m. plus 45 seconds on day number 234. The following program sets the interrupt time and stores in the clock the instructions to issue on interrupt.

```
10 SERVICE 9
20 OUTPUT (9,40)"INT",234,15,30,45,0
30 OUTPUT (9,40)"KEY",24,49,50,44,49,48,44,49,48,11
40 FORMAT 10F1000.0
```

Line 20 sets the interrupt time to Julian Day 234, 3:30 p.m. plus 45 seconds. Line 30 stores the instruction LOAD 12,10,10, execute. Refer to Appendix B where the key values are given and note the correlation. The first number (24) is the LOAD key and the following keys are 1,2 , , , 1,0 , , , 1,0, EXECUTE. The instruction uses 10 of the 32 keystrokes of instructions that the RT-30 can store. Note that LOAD, a four character word, is still only 1 keystroke and uses only 1 of the 32 keystrokes of the RT-30 memory capacity.

As a part of its operation, the program on file 12 could read the time and compute a new interrupt time and store a completely new set of instructions in the clock.

THE KEY FUNCTION

The KEY function returns the decimal value of the last key pressed. The function provides a mask expression so that its power and flexibility is substantially enhanced. The function uses 1 word of read-write memory as a last key register where it temporarily stores each keystroke. When the last key register is interrogated, its contents are replaced by a −1, indicating that a keystroke is no longer stored. The logical operation of KEY and the decimal values of the 9830 keys are given in Appendix B.

SYNTAX:

KEY (mask expression)

Expression, if zero is a means for the programmer to accomplish what is, in effect an RBYTE*

A. If the last key register is not equal to −1, then the contents of this register are returned in both the result register and as a result of the function. The last key register will then be set to −1.

B. If the last key register is −1, then the function will wait for a key to be pressed or until it is aborted by the STOP key. If the function is aborted, the last key register will remain unchanged. If the machine is in the UNBREAK mode provided in the Fast Basic II ROM, an abort by the STOP key is not permissible and if attempted, will result in loss of machine control. This is consistent with the requirement that UNBREAK prevents tampering from the keyboard.

*See the RBYTE operation in the Extended I/O manual, page 3-4.

Expression, if not zero, is a mode of the KEY function that provides a means of testing for keyboard activity and returns a result immediately regardless of value. In this mode, KEY takes the contents of the last key register, copies the value to the result register, and returns the masked value as a result of the function. The last key register is set to −1 upon exiting the function.

SYNTAX EXAMPLE:

```
KEY(0)
```

The mask value Ø will cause the function to stall program execution until a key is depressed. All keys except STOP, REWIND, and SHIFT will be recognized instantly and the decimal value returned.

USAGE EXAMPLE:

```
10 DISP KEY(0)
20 WAIT 1E+03
30 GOTO 10
```

THE KEY FUNCTION [Cont.]

This three line routine returns the decimal value of the last key stroke. After pressing RUN EXECUTE, an 11, the value of the EXECUTE key, will be displayed until a different key is pressed. After the first pass, the program will hold at line 10. When any key other than STOP, REWIND, or SHIFT is pressed, the function will return the decimal value of the key pressed.

USAGE EXAMPLE:

Let us assume that it is possible for a program to run for an unknown length of time which could be many hours. Because a more urgently required program must be run, or it is time to shut-down for the night, it may be desirable to suspend operation. The following program demonstrates how KEY may be used to interrogate the operation of a program to determine its completion status without halting the program. This is crucial as a halt would destroy for-next loop and gosub return data.

```
10 MDUMP -1
20 A=KEY(-1)
30 M=RND(1)*3E+04
40 FOR N=1 TO M
 .
 .
 .
1000 IF KEY(-1)#77 AND RES#78 THEN 1060
1010 GOTO RES-76 OF 1020,1040
1020 DISP M
1030 GOTO 1050
1040 DISP N
1050 WAIT 1E+03
1060 NEXT N
1070 END
```

This program demonstrates how the user can determine the completion status of a program of indefinite running time. He can test to see how long it takes the for-next loop of the program to go one cycle by interrogating N at a known interval. Then he can test the value of M to determine how many times the loop must be executed. From this he can determine whether he wishes to wait for the program to finish or suspend it. As the program has a for-next loop, stopping the program to make such a determination would destroy the for-next loop counter. This example demonstrates that KEY is a particularly powerful complement to the MDUMP feature of Fast Basic III.

Line 10 puts the machine in the MDUMP (via keyboard) mode. Line 20 sets the last key register to a value of −1 so that the last key pressed initiating program execution will not have any effect. Line 30 initializes the variable M to a random number and line 40 begins a for-next loop to the random number M. At line 100 the KEY function tests to see if a key was pressed. The result register is used to avoid a second function call because the function has already reset the last key register to − 1. It is important that the KEY function be evaluated PRIOR to the RESULT register reference. Also, line

THE KEY FUNCTION [Cont.]

1000 tests to see if either an N or an M has been pressed.

If not, the loop continues, if true, execution is branched to line 1010. At line 1010 a computed go-to branches execution to either line 1020 or 1040 depending whether an M or N was pressed. In either case, the value is displayed for 1 second and execution continues.

NOTE FOR HP MASS MEMORY USERS

The HP Mass Memory system has an anomaly which will erroneously issue an error 93 when an attempt is made to enter an IF statement immediately followed by KEY. For example, the following statements will produce an error 93:

```
250 IF KEY(-1)<0 THEN 300
```

This anomaly may be circumvented by transposing the statement as follows:

```
250 IF 0>=KEY(-1) THEN 300
```

The Infotek Mass Memory II systems bootstraps eliminate this anomaly. When used, no special treatment need be given the IF statement.

NOTE: If the LOWCASE or UNBREAK mode is active, the key function is disabled.

NUM STATEMENT

The NUM statement returns the decimal ASCII value of a specified character in a string. This is particularly useful in unpacking data from strings as the value that can be assigned to a single byte (or string character) may be any number between Ø and 225. Most numbers between Ø and 255 can be generated by the keys on the 9830. Each key, as shown in Appendix B, has a specific shifted and unshifted decimal value. Any number between Ø and 255 can be output to a string character by the Extended I/O ROM. The String ROM is required. Error 41 will result if the character is null.

SYNTAX:

NUM (String or Substring)

String Variable — can be any previously defined string or substring.

EXAMPLE:

```
V=NUM(A$[3])
```

In the example the variable V takes on the decimal value of the third character of A$.

USAGE EXAMPLES:

```
10 DIM A$[3]
20 DISP "INPUT: MONTH, DAY, YEAR";
30 INPUT M,D,Y
40 OUTPUT (A$,50)M,D,Y;
50 FORMAT 3B
60 PRINT NUM(A$);NUM(A$[2,2]);NUM(A$[3])
70 PRINT
80 END
```

The preceding program demonstrates a means whereby 3 two digit numbers can be packed into 1-½ words (3 bytes) of memory. In this example, the month, day, and year are stored in 3 consecutive characters. The value of M, D, and Y are output to A$ in line 40. Line 60 retrieves the values by use of the NUM statement.

The TRANSFER statement of APII and the DBYTE statement of the Mass Memory system provide additional means of placing any 8 bit value in a string character.

NUM has particular application in the packing of information in the 9880B Mass Memory files wherein disc space is allocated on the basis of 1 byte per string character or 4 bytes per integer precision element. Accordingly, it is far more efficient to store information in the form of two 255 character strings rather than one 128 element integer precision array. The two strings could hold slightly over 500 numbers, whereas the integer precision array could hold a maximum of 128 numbers. NUM provides the vehicle for conveniently retrieving information packed into strings.

NUM STATEMENT [Cont.]

DBYTE METHOD:

```
10 DIM A$[3],I$[1]
20 DISP "MONTH, DAY, YEAR (MM,DD,YY)";
30 INPUT M,D,Y
40 DBYTEM,A$
50 DBYTED,I$
60 A$[2]=I$
70 DBYTEY,I$
80 A$[3]=I$
90 PRINT NUM(A$);NUM(A$[2]);NUM(A$[3])
100 END
```

TRANSFER METHOD:

```
10 DIM A$[3],D[3]
20 DISP "MONTH, DAY, YEAR (MM,DD,YY)";
30 INPUT D[1],D[2],D[3]
40 D[1]=D[1]*256+D[2]
50 D[2]=D[3]*256
60 TRANSFER D[1] TO A$
70 PRINT NUM(A$);NUM(A$[2]);NUM(A$[3])
80 END
```

UCASE STATEMENT

The UCASE statement scans the contents of a string for lower case characters and, if present, converts them to their upper case equivalents. UCASE is particularly useful in guarding against inadvertent operator entries of lower case characters which would result in improper alphabetic sorting. The String ROM is required for all string operations.

SYNTAX

UCASE string or substring

The string must be initialized and substring references if used must be within the bounds of the string.

SYNTAX EXAMPLE:

```
UCASE A$
```

Any lower case characters in A$ will be converted to the corresponding upper case characters.

USAGE EXAMPLE:

```
10 DIM A$[32],A1[10,17]
20 FOR N=1 TO 10
30 DISP "ENTER NAME";
40 INPUT A$
50 UCASE A$
60 TRANSFER A$ TO A1[N,2]
70 A1[N,1]=N
80 NEXT N
90 END
```

The example shown above anticipates that the names stored in the array will require sorting alphabetically. To ensure that lower case characters are eliminated (which have numberic value greater than upper-case) each entry is converted to upper-case before storing in the array. This is accomplished at line 50.

SHF FUNCTION

The SHF function allows any integer value (16 bit number) to be shifted from 1 to 16 bits to the right or left. The Extended I/O ROM is required.

SYNTAX:

SHF (integer expression, + or − number of bits)
Expression — an integer value to be shifted
+ — indicates shift right.
− — indicates shift left.
Number of Bits — Number of bits to shift (may be an expression).

SHIFT differs from rotation. In a SHIFT function, bits shifted beyond the 16 bit bounds of the word are discarded rather than "wrapped around" to the other end of the word.

EXAMPLE:

```
10 A=256
20 PRINT A
30 FOR I=1 TO 8
40 PRINT SHF(A,I)
50 NEXT I
60 PRINT
70 END

 256
 128
 64
 32
 16
 8
 4
 2
 1
```

Line 40 shifts A, 1 number of places to the right and prints the result. Numbers greater than integer (±32,767) are treated as −32,768.

CMP FUNCTION

The CMP function returns a 1's complement of an integer precision number.

SYNTAX:

CMP (integer expression)

The expression must be reduced to an integer value.

SYNTAX EXAMPLE:

```
CMP(1)
```

The function will return the 1's complement of 1 which is −2.

CERROR STATEMENT

CERROR cancels a previously executed SERROR statement. It is extremely useful in complex programs undergoing debugging where SERROR is desired in debugged areas and not desired in other specific areas. The APII ROM is required.

SYNTAX:

CERROR

```
CERROR
```

When executed, CERROR clears the most recently executed SERROR. To return to program controlled error recovery, another SERROR statement must be executed.

RXREF COMMAND

This command produces a list of ROMs required to run a program. Optionally, all program lines that use a specific ROM can be listed. RXREF requires the Infotek Fast Basic I and II ROMs. Also, either the Infotek Mass Memory II, in conjunction with the HP 9880B, or the Infotek RXREF binary tape is required.

SYNTAX:

RXREF [# select code [, ROM number]]

Select Code — any select code between 1 and 9, or 15 is allowed. If a number is not specified, a default of 15 is used.

ROM Number — Optional — If not specified, the command produces a listing of ROMs used by the program against the line numbers that use each ROM.

If Specified — the command produces a listing of all program lines which use the ROM. When the optional specification is used, the output device select code must be specified even if select code 15 is desired.

USAGE EXAMPLE:

It is desired to quickly find all references to flags in a very long program where flags are frequently submerged in complex arguments. A task which, if performed manually, would be tedious and susceptible to omissions.

The following command will list all lines which use the APII ROM, thereby greatly reducing the effort required to find flag references:

```
RXREF #15,15
```

As 15 is the code number assigned to the APII ROM, all lines using APII will be listed.

USAGE EXAMPLE:

Upon loading a program, an Error 1 is obtained. Obviously a ROM required by the program is not installed. To find out which ROM is missing, RXREF is used. On completion of the RXREF command, one needs only to list the program and note the line numbers in which the Error 1 message is obtained and find that number in the RXREF list. The ROM number printed next to the Error 1 line number is missing.

RXREF COMMAND [Cont.]

The following is an RXREF of the usage example program on page 8:

```
RXREF
5       10    20    50    70    80    100   120   130   140   160   170
6       30
15      150
22      60    90    110
23      40
```

An RXREF#15,22 produces a listing of lines which reference Fast Basic III instructions. Using the example program on page 8, the following listing results.

```
RXREF#15,22
60 VDIM B[1000,4]
90 VSEARCH B,C,1,P,X
110 ARRAY B[X,2],0

25
```

ROM NAME TO ROM NUMBER CORRELATION

1		17	
2	Strings	18	
3	Matrix Operations	19	
4	Plotter	20	
5	Basic (language)	21	
6	Basic (cassette)	22	Fast Basic III
7	Terminal I/DataCom III	23	Fast Basic I and II
8	Mass Memory/MMII	24	
9	API	25	SCAT
10	Batch Basic	26	
11	Extended I/O	27	
12		28	
13	DataComm I	29	
14	DataComm II	30	
15	APII	31	
16	Printer Control/Typewriter	32	

NOTE: 24 thru 28 may be used by binary programs.

As new ROMs are developed, their number can be determined by RXREF. Blank spaces have been left in the above list so the user may note new ROM assignments. A blank space indicates that at time of publication of this manual, no assignment for that ROM number was known to INFOTEK.

APPENDIX A

NAMES LIST

VDIM

VDIM array name (rows[, columns])

Allows any array to take on virtual dimensions which may be greater than 256 elements on a side.

VSORT

VSORT array name, {R or C},#[,#[,#[,#[,#[,#]]]]]

Permits sorting arrays that are virtually dimensioned.

VSEARCH

VSEARCH array name, {R or C}, #, match expression, return variable

Allows searching arrays that are virtually dimensioned.

ARRAY

ARRAY array (subscripts), return variable[, expression]

Allows virtually dimensioned array elements to be loaded or stored according to their virtual subscripts.

VROW

VCOL

VROW (array name)

VCOL (array name)

Returns the number of rows and columns in virtually dimensioned arrays.

VPRINT#

VREAD#

VPRINT# file no. [,record no.]; array name

VREAD# file no. [, record no.]; array name

Allows virtually dimensioned arrays to be printed on or read from the HP 9880B Mass Memory system.

MREAD

MWRITE

MREAD (select code, {1 or 2} , byte count expression) array (subscripts)

MWRITE (select code, {1 or 2} , byte count expression) array (subscripts)

Allows data to be input or output from the 9830 at rates in excess of 10,000 bytes per second.

MDUMP

MDUMP enable/disable mode expression

Provides a means of suspending a running program to cassette or floppy at any time for later completion.

APPENDIX A [Cont.]

SERVICE

SERVICE select code

Allows a peripheral device to interrupt the 9830 and issue instructions via the I/O bus which are interpreted as keystrokes on the keyboard.

KEY

KEY (mask expression)

Returns the decimal value of the last keystroke.

NUM

NUM (string or substring)

Returns the decimal value of a string character.

UCASE

UCASE string or substring

Converts any lowercase characters of a string to the uppercase equivalent.

SHF

SHF (integer expression, + or − number of bits)

Shifts an integer number the specified number of bits left or right.

CMP

CMP (integer expression)

Returns a 1's complement of the specified number.

CERROR

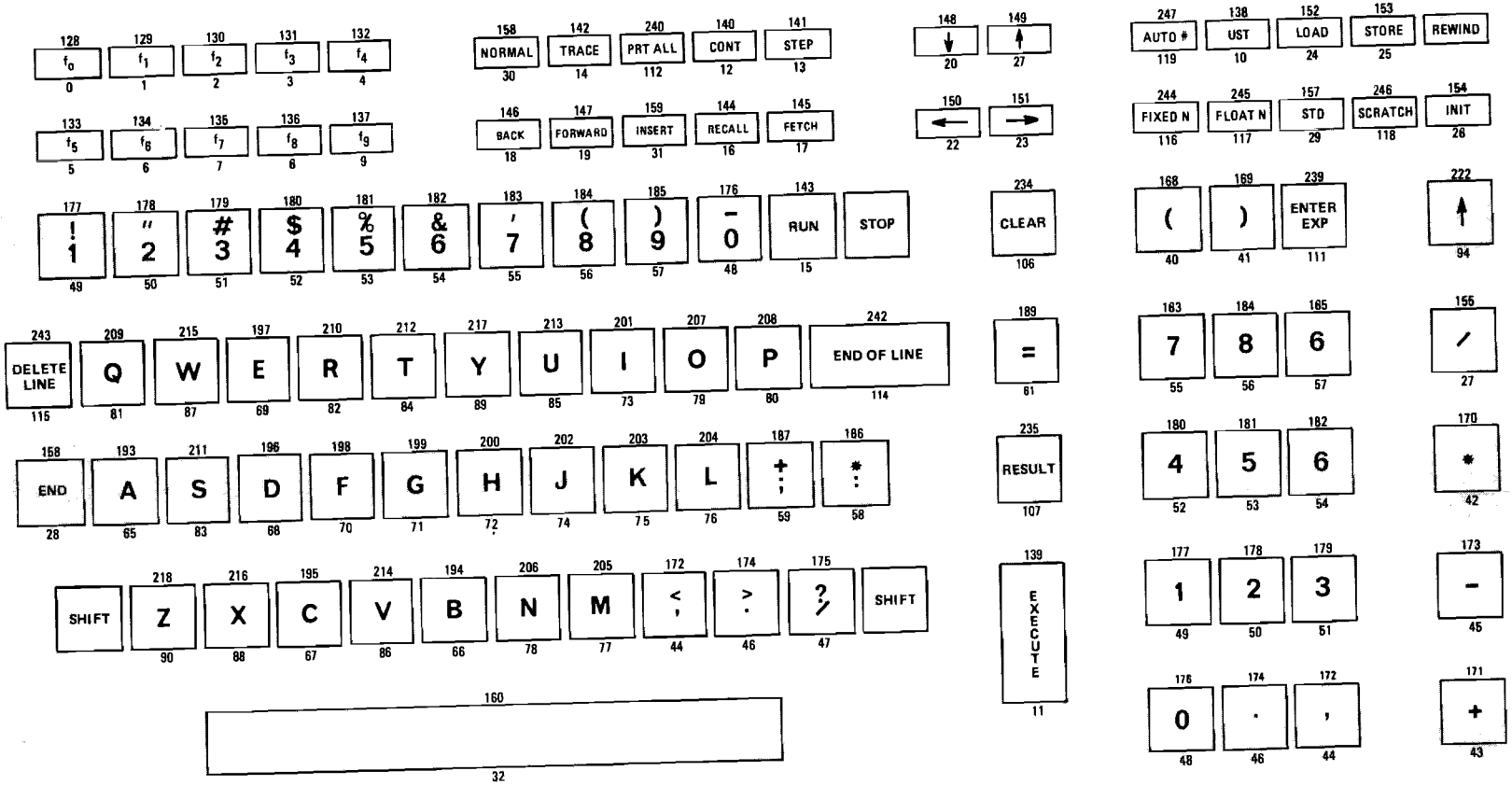
CERROR

Clears the SET error statement.

RXREF

RXREF [# select code[, ROM number]]

Provides a listing of ROMs used by a program against the line numbers in which the ROMs are used; also, a list of all lines which use a specific ROM.



NOTE:
THE NUMBER ABOVE KEY IS THE SHIFTED VALUE
ALL NUMBERS ARE DECIMAL VALUES

APPENDIX B1

DECIMAL CODES OF THE KEYBOARD

APPENDIX B2

KEYBOARD CODE TABLE

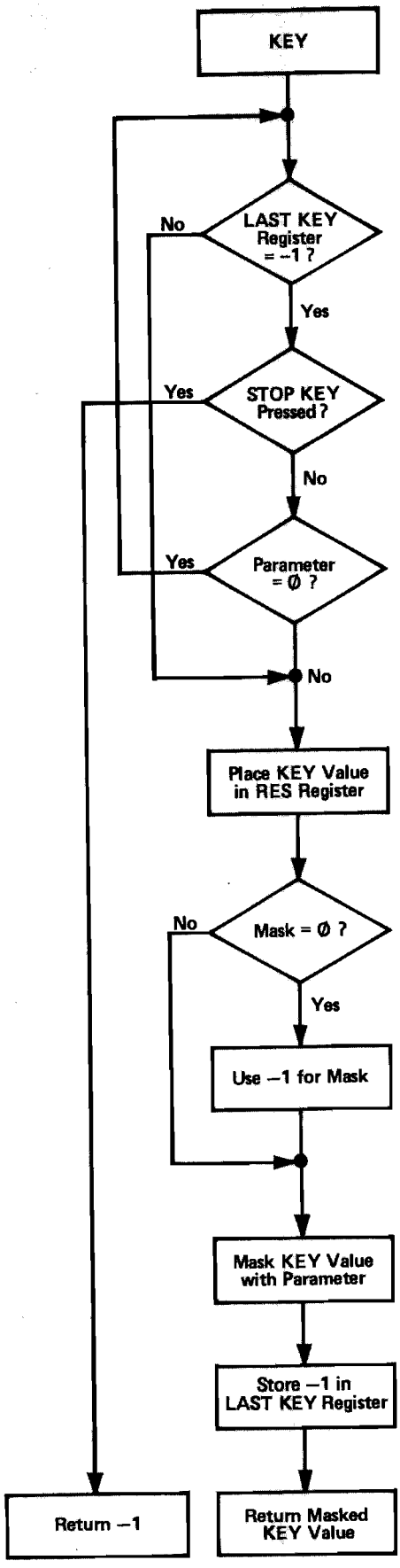
DECIMAL CODE	KEY (or SYMBOL)	DECIMAL CODE	KEY (or SYMBOL)	DECIMAL CODE	KEY (or SYMBOL)
0	f_0	46	.	92	▒
1	f_1	47	/	93	]
2	f_2	48	0	94	↑ (raise to power)
3	f_3	49	1	95	⊢
4	f_4	50	2	96	@
5	f_5	51	3	97	shift A
6	f_6	52	4	98	shift B
7	f_7	53	5	99	shift C
8	f_8	54	6	100	shift D
9	f_9	55	7	101	shift E
10	LIST	56	8	102	shift F
11	EXECUTE	57	9	103	shift G
12	CONT	58	:	104	shift H
13	STEP	59	;	105	shift I
14	TRACE	60	<	106	CLEAR
15	RUN	61	=	107	RES
16	RECALL	62	>	108	shift L
17	FETCH	63	?	109	shift M
18	BACK	64	@	110	shift N
19	↓ FORWARD	65	A	111	E (exponent)
20	↑ (edit keys)	66	B	112	PNT ALL
21	←	67	C	113	shift Q
22	→	68	D	114	END OF LINE
23		69	E	115	DEL LINE
24	LOAD	70	F	116	FIXED N
25	STORE	71	G	117	FLOAT N
26	INIT	72	H	118	SCRATCH
27	/	73	I	119	AUTO #
28	END	74	J	120	shift X
29	STANDARD	75	K	121	shift Y
30	NORMAL	76	L	122	shift Z
31	INSERT	77	M	123	[
32	Space Bar	78	N	124	▒
33	!	79	O	125	]
34	"	80	P	126	↑ (raise to power)
35	#	81	Q	127	STOP
36	$	82	R	128	shift f_0
37	%	83	S	129	shift f1
38	&	84	T	130	shift f2
39	'(apos.)	85	U	131	shift f3
40	(	86	V	132	shift f4
41	)	87	W	133	shift f5
42	*	88	X	134	shift f6
43	+	89	Y	135	shift f7
44	,(comma)	90	Z	136	shift f8
45	–	91	[	137	shift f9

APPENDIX B2 [Cont.]

DECIMAL CODE	KEY (or SYMBOL)
138	LIST
139	EXECUTE
140	CONT
141	STEP
142	TRACE
143	RUN
144	RECALL
145	FETCH
146	BACK
147	FORWARD
148	↓
149	↑ (edit keys)
150	←
151	→
152	LOAD
153	STORE
154	INIT
155	/
156	END
157	STANDARD
158	NORMAL
159	shift INSERT (character delete)
160	Space Bar
161	!
162	"
163	#
164	$
165	%
166	&
167	'(apos.)
168	(
169	)
170	*
171	+
172	<
173	–
174	>
175	?
176	–
177	!
178	"
179	#
180	$
181	%
182	&
183	'(apos.)
184	(
185	)
186	*
187	+
188	<
189	=
190	>
191	?
192	@
193	shift A
194	shift B
195	shift C
196	shift D
197	shift E
198	shift F
199	shift G
200	shift H
201	shift I
202	shift J
203	shift K
204	shift L
205	shift M
206	shift N
207	shift O
208	shift P
209	shift Q
210	shift R
211	shift S
212	shift T
213	shift U
214	shift V
215	shift W
216	shift X
217	shift Y
218	shift Z
219	[
220	▒
221	]
222	↑ (raise to power)
223	⊢
224	@
225	shift A
226	shift B
227	shift C
228	shift D
229	shift E
230	shift F
231	shift G
232	shift H
233	shift I
234	CLEAR
235	@
236	shift L
237	shift M
238	shift N
239	E (exponent)
240	PRT ALL
241	shift Q
242	END OF LINE
243	DEL LINE
244	FIXED N
245	FLOAT N
246	SCRATCH
247	AUTO #
248	shift X
249	shift Y
250	shift Z
251	[
252	▒
253	]
254	↑ (raise to power)
255	STOP

APPENDIX B3

FLOWCHART OF THE KEY FUNCTION



APPENDIX C

CROSS INDEX OF ROMs REFERENCED BY FAST BASIC III

	MATRIX	STRING	EXTENDED I/O	APII	FBI	FBII	MMII	EXECUTION ERROR MESSAGES GENERATED
VDIM	X							41, 67
VSORT				X				37, 38, 41
VSEARCH				X				37, 38, 41
ARRAY	X							37, 38, 40, 41, 42, 105
VROW								37, 38, 41
VCOL								37, 38, 41
VPRINT	X						X	37, 38, 41, 94
VREAD	X						X	37, 38, 41, 94
MREAD					X			43, 61, 205
MWRITE					X			43, 61, 205
MDUMP								
SERVICE								43
KEY								
NUM		X						72
UCASE		X						
SHF			X					
CMP								
CERROR								
RXREF						X	X*	94

*Binary Tape may be substituted.

NOTES

INFOTEK FAST BASIC III ROM

INSTRUCTION MANUAL

Infotek Systems

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

PL1M7709 PRINTED IN U.S.A.