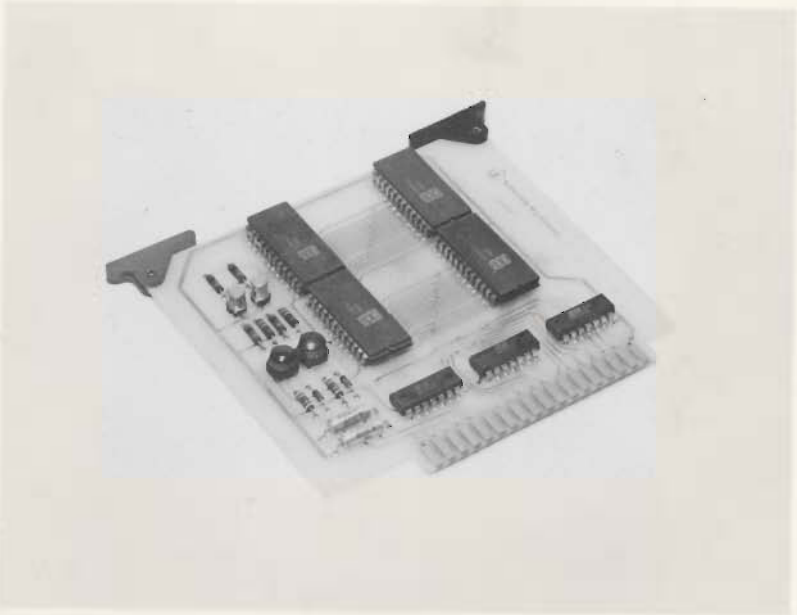
FAST BASIC II ROM
FOR THE HEWLETT-PACKARD 9830A/B COMPUTER

INSTRUCTION MANUAL

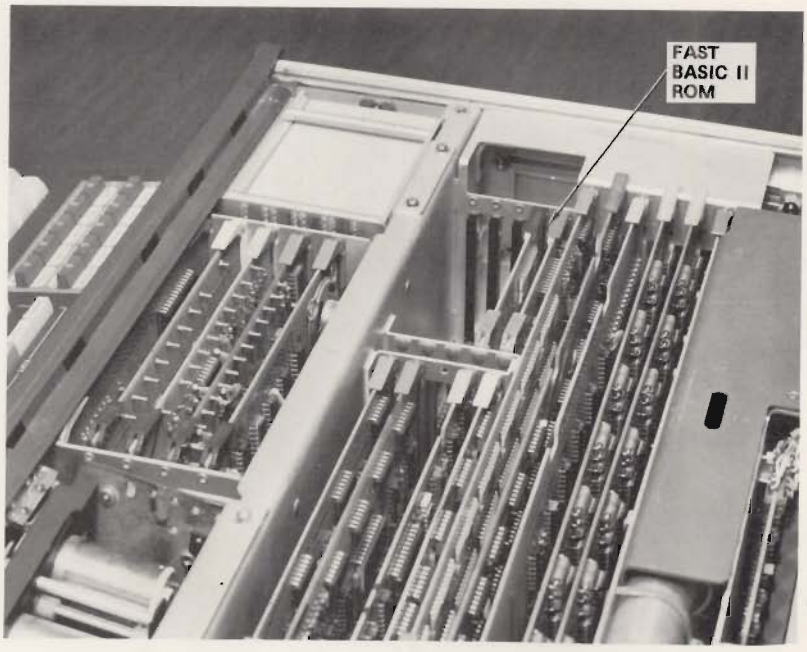
Infotek Systems

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

INFOTEK FAST BASIC II ROM



FAST BASIC II ROM
Circuit Card



FAST BASIC II ROM
Circuit Card Installed
in HP 9830A/B Computer

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

INSTRUCTION MANUAL

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

Computer Museum

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 II ROM is an extension of FAST BASIC I and functions only in conjunction with FAST BASIC I. Its primary purpose is to provide a universal means of tape file management. In this respect, FAST BASIC II allows the duplication of data or program files from any device select code to any other device select code. Moreover, FAST BASIC II allows redefinition of file headers to circumvent the 50% slack tape constant of the 9830 cassette drive. By packing up to 50% more data on a cassette, the capacity and effective speed of the tape system is increased. FAST BASIC II permits manipulation of program files by a basic program. This makes it possible to implement utility programs which can duplicate, and in the process, if desired, can also compress files on tape.

Another important capability made possible by FAST BASIC II is the application of Infotek text editing utilities to programs. For example, if it becomes necessary to insert a #5 in all cassette commands, a text editing program does this. Changing select code numbers in WRITE statements and re-defining TRANSFER and BEEP statements from API TO APII are also accomplished by text editing utility programs available from Infotek.

A secondary, but equally important, purpose of FAST BASIC II is to provide an effective **ninth** ROM capability for the 9830. In addition to its unique repertoire, FAST BASIC II finishes the duplication of API functions not covered by APII, so that more efficient use can be made of the eight available OPTIONAL ROM slots. FAST BASIC II **does not use an optional ROM slot.** It replaces an existing non-optional ROM inside the machine.

While FAST BASIC II does not change any of the existing 9830 functions, it does expand the machine language by the ten statements, functions and commands listed. In applications where FAST BASIC I and FAST BASIC II are installed in lieu of API there is a **NET GAIN** of 31 statements, functions and commands beyond the capabilities of a 9830 with the API ROM. SCROLL and DFC are the only API functions deleted.

SYNTAX CONVENTIONS

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

INSTALLATION

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 holddown brackets (Figure 2). Note the location of the brackets and how they are attached. Remove the brackets.

7. Only the fourth card position behind the front panel along the left side of the computer may be used for FAST BASIC II (see Figure 3). The standard ROM located in this slot is identified by a blue handle on the left side of the card and a red handle on the right side of the card. Remove this card as it is **REPLACED** by FAST BASIC II.

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 power 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 II 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.

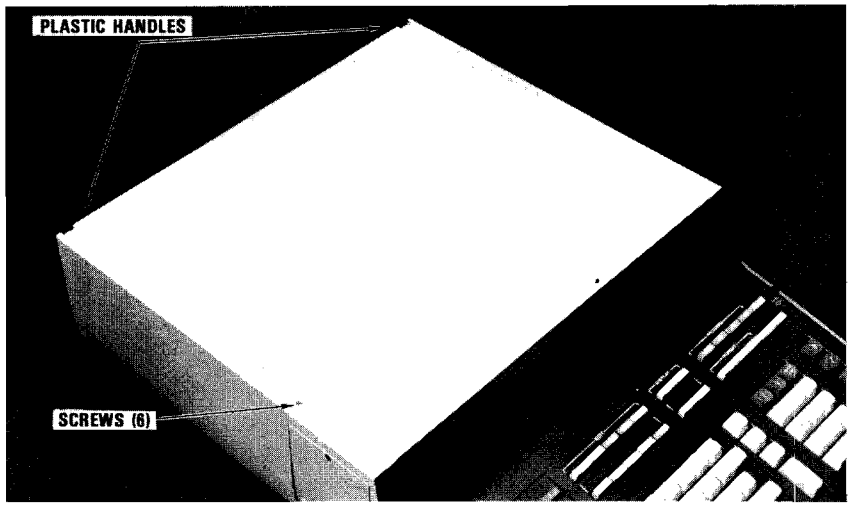


Figure 1.
Removal of Cover

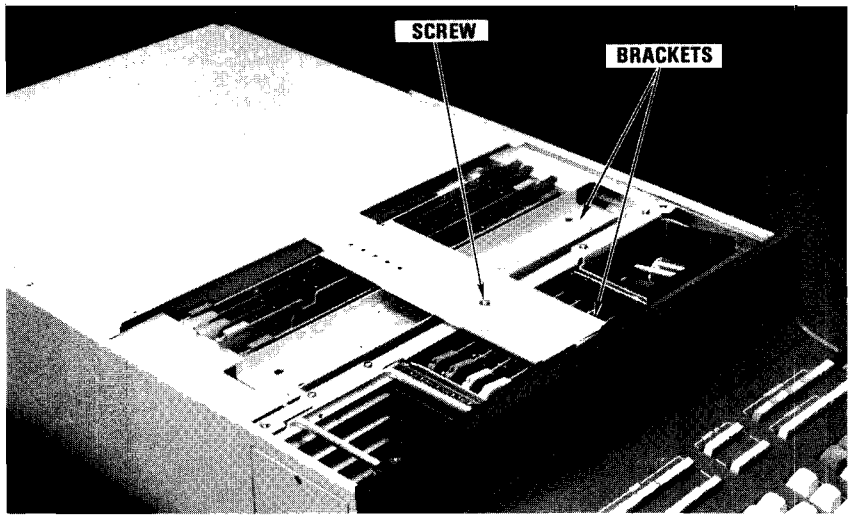


Figure 2.
Removal of Brackets

Figure 3.
Location of
FAST BASIC II
Circuit Card

FDATA STATEMENT

FDATA L allows both data files and program files to be loaded from tape to a specified array. FDATA S is the complement of FDATA L, providing a means to store data arrays as program files or as data of a different precision. The FDATA statements provide universal files management capability under program control. Among other benefits, this permits packing 50% more information on a tape and, in effect, increasing the cassette system speed by 50% as well. Also, because programs can be treated as arrays, Infotek text editing utilities can be used to re-define duplicate ROM statements and change select codes.

Typical application of utility programming made possible by using the FDATA statement is moving data and program files from one peripheral tape or floppy disk to another. This permits select code specification for both the source and destination devices. While moving the files, a program could also re-define TRANSFER and BEEP statements from API to APII and insert #5 in all cassette control statements. The select codes of write statements could be changed selectively and excess file space could be removed from the cassette tape in the process. Moreover, multiple source tapes or disks can be edited and stored on a single destination tape or disk.

In conjunction with the EXOR statement of FAST BASIC I, both data and program files can be super-secured making magnetic media useless to unauthorized persons. As 65,534 code keys are possible and machine control is lost each time an incorrect code key is used on a program, the information is indeed highly secure.

Refer to the appendices for specific examples.

SYNTAX:

FDATA [#select code ,] S , file header array (subscripts) , file data array
FDATA [#select code ,] L , file data array

EXAMPLES:

```
FDATA S,H(1),A
FDATA #5,S, H(1),A
FDATA L,A
FDATA #5,L,A
```

where A is an integer precision array into which data is to be loaded, or from which data is to be stored and H is an integer precision array containing file header information. The select code specification is optional and defaults to select code 10 if not used.

FDATA L

Before FDATA L may be used, a FILEID statement should be executed. FILEID brings into memory all pertinent information regarding the file to be loaded. If the file to be loaded is not the next file in order, FIND must be executed to position the tape at the desired file. Consideration must be given to the fact that the FIND statement returns control to the processor upon encountering the first file following the command. In circumstances where this is not the desired file, the wrong file will be loaded. Accordingly, the FILEID statement, the WAIT statement, or both in combination, should be used to assure that the tape is positioned at the desired file (see the program in Appendix A, figure A1).

FDATA L may be used to load any type of file except binary (type 1, or 21). ERROR 62 results if an attempt is made to use FDATA L to load a binary file.

The array into which data is to be loaded must be of sufficient dimensions to hold all the data in the file. The array must have at least the same number of words as the current file size. ERROR 88 results if the array is smaller than the current file size. ERROR 59 may result if an attempt is made to load an empty file with FDATA L.

This statement assumes that the tape is positioned between the file header and file body upon execution; i.e., a FILEID had previously been executed. If the tape is otherwise positioned between files then the previous file will be loaded instead of the next file.

FDATA STATEMENT [Cont.]

FDATA S

FDATA S may be used to store any type of file, since the file type is specified by the user in the file header array. However, the file number specified in the header array must be the same as that of the file header on the tape or disk. For example, if data is to be stored on file 12, the file header array must specify 12 for the file number. ERROR 62 results if the actual file number and the new file header do not correspond. The file header array must be an integer array with at least 7 locations. It should contain the following information:

Location	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 (for program files)
7	Common area in words (for program files)

FDATA S will store only as much data from the array as is specified by the current file size in the file header array. ERROR 62 will occur if the current file size exceeds the absolute size specified in the header array.

Since the user may specify the file type to be used when storing data on a tape or disk with FDATA S, it is possible to create entirely new file types. In addition, the three words in the file header which are normally used by the 9830 for system information (words 5, 6 and 7) may be used by the programmer in any fashion desired, so long as the file is loaded by means of FDATA L. Identification of data which would normally be stored in a file and would require reading of the entire file may now be obtained simply by reading the file header with FILEID, or by doing a TLIST of the tape or disk. A program may use such file header locations to identify the file as a part of a given software system.

FDATA S [Cont.]

When creating new file types avoid certain file type numbers reserved for use by different elements of the 9800 computer series. The following table lists types currently in use.

Type	Use
0	Empty file
1	Binary file
2	Data file
3	Program file (BASIC)
4	Key file
5	Special program file
7	Fast Basic III memory dump file
10	9810A program file
11	Secure 9810A program file
20	HP 9820A/9821A program file
21	Secure binary file
23	Secure BASIC program
24	Secure KEY file
25	Secure special HP program file
28	HP 9820A/9821A special program file
58	Mass Memory bootstrap file

Any integer number may be used to define a data file type. However, it must be remembered that only the two most significant digits will be presented by a TLIST. The FILEID, of course, will return the entire number.

FDATA STATEMENT [Cont.]

FDATA S [Cont.]

Additional Rules for Using FDATA S:

1. This statement assumes that the tape is positioned between files.

2. There must be at least 12 words of space from the beginning of the file header array to the end of memory in a 16,096 word memory 9830 or ERROR 88 will be issued for attempted memory wrap around.

3. If the file header array is less than 7 words long ERROR 62 will be issued. Only the first seven words beginning with the element specified in the statement will be written into the tape file header.

4. If the current file size word (#2) in the header array is bigger than physical memory would permit then ERROR 88 will be issued.

5. If the current file size word (#2) in the header array is bigger than the current working size of the data array then ERROR 62 will be issued.

6. If the current file size word (#2) in the header array is negative then ERROR 62 will be issued.

7. If the absolute file size word (#4) in the header array is negative then ERROR 62 will be issued.

8. The absolute file size may be reduced or increased by the user by way of FDATA S. However, the absolute file size word (#4) must not be less than the current file size word (#2) in the header array. If the current file size word is zero (Ø) then the absolute file size word may be any positive value. Otherwise, ERROR 62 will be issued.

9. The absolute file size word (#4) in the header array cannot be greater than the actual absolute file size or ERROR 62 will be issued unless the current file size word (#2) is zero (Ø).

CODE STATEMENT

The CODE statement is a multi-purpose statement which performs five different functions defined by a numeric expression. Two of these functions are used to control flags in conjunction with APII. The other three control the operating mode of the 9830.

SYNTAX:

CODE n where n is an integer expression between -1 and 18.

CODE 0 THRU 15

When n is 0 through 15, the corresponding APII flag will be complemented (i.e. if it is cleared, it will be set; if set, it will be cleared). These codes have no meaning and will produce ERROR 1 without the Advanced Programming II ROM installed.

CODE 16

Clears all 16 flags of APII. This code has no meaning and will produce ERROR 1 without the Advanced Programming II ROM installed.

CODE 17

The 9830 will enter LOWCASE mode; all characters will be entered in lower case **UNLESS** the SHIFT key is pressed. This is the complement of the normal mode of the machine, where uppercase characters are entered **UNLESS** the SHIFT key is pressed. The lowcase as implemented in FAST BASIC II does not have the Advanced Programming I anomaly of causing a SCRATCHALL if a FIND command is executed in the LOWCASE mode. It is not necessary to jump from lowcase to uppercase whenever a FIND must be executed. CODE -1 resets lowcase to the normal HIGHCASE mode.

CODE 18

This code puts the computer in UNBREAK mode. In this mode, the STOP key has no effect during program execution. This prevents tampering with the machine while running programs which have long iterative processes. The UNBREAK mode remains in effect until reset by a CODE -1. Neither the LOW CASE or UNBREAK mode can be initiated unless the computer is in the "normal" mode.

CODE -1

CODE 17 and CODE 18 are both cancelled if n in the CODE n statement is negative. The LOWCASE and UNBREAK modes are mutually exclusive and cannot be in effect concurrently. The CODE statement executed first determines the mode of the computer.

SUMMARY OF CODE STATEMENT FUNCTIONS

CODES 0 thru 15	Complements the specified flag
CODE 16	Zeroes all 16 flags at once
CODE 17	Puts keyboard in LOWCASE mode
CODE 18	Sets UNBREAK mode
CODE -n (any negative number)	Puts keyboard in HIGHCASE or resets UNBREAK ("normal" mode)

CAUTION

Since the only way to stop a program when the computer is in the UNBREAK mode is to turn off the computer, it is recommended that use of CODE 18 be limited to thoroughly proven programs.

POWER-ON AUTO-START

FAST BASIC II provides the 9830 user with the capability of loading and executing a program from cassette or Mass Memory automatically upon power application.

The logic of the power-up routine provides substantial flexibility. If a Mass Memory ROM is installed in the computer, FAST BASIC II will attempt to load and run a program called "↑" (up-arrow) from unit Ø. If the Mass Memory is not ready or the power is off, the 9830 will beep once every two seconds until the drive is ready. If an auto-start is not desired, simply press the STOP key.

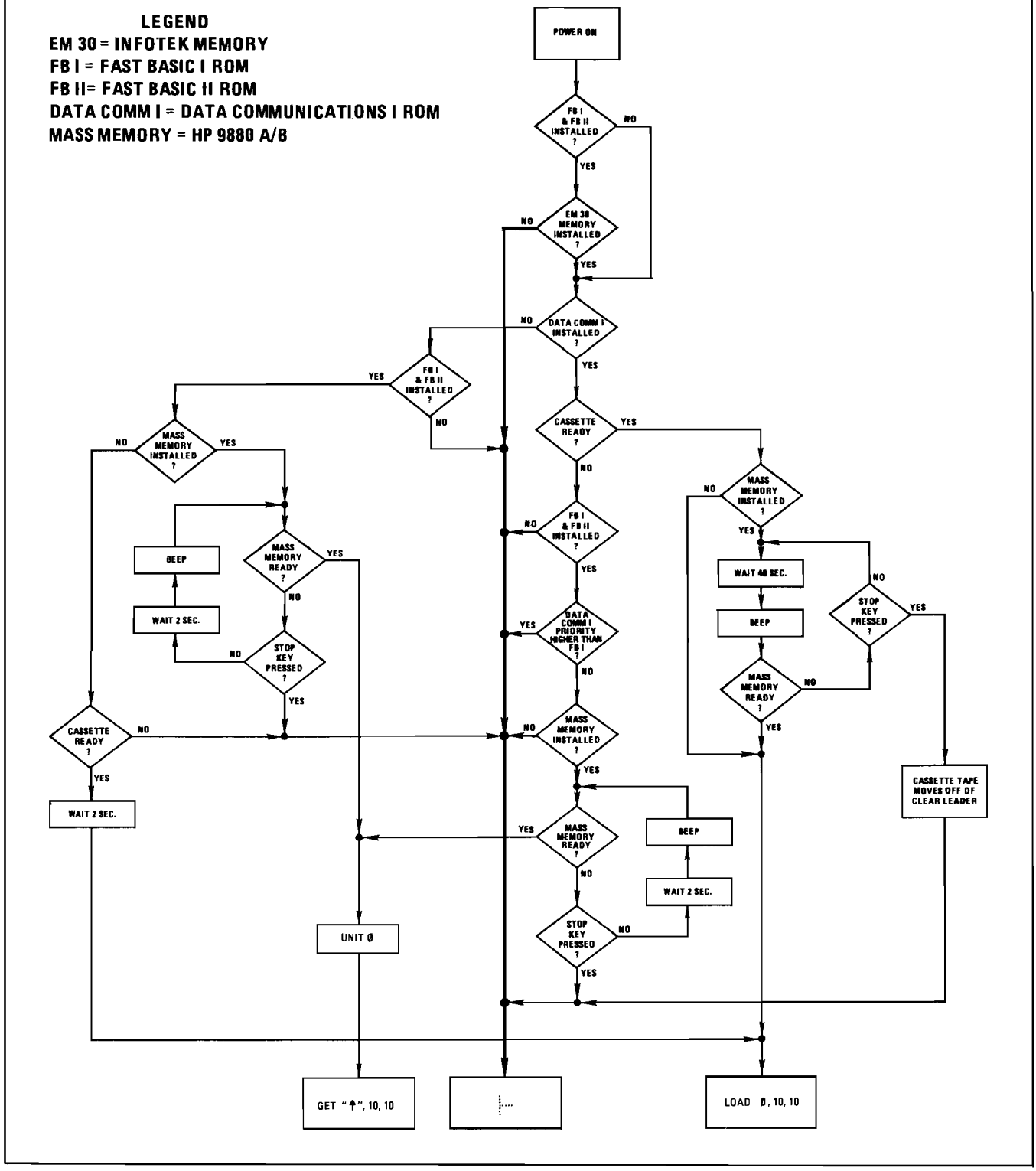
If a Mass Memory ROM is not installed, the 9830 will attempt to execute a LOAD Ø, 10, 10 command. If the internal cassette is not ready, the computer will display a lazy T after a two-second wait.

If the 9830 is also equipped with the Data Communications I ROM (which has a cassette power-on auto-start capability) then the relative priority of it to the FAST BASIC I ROM affects the operation of the power-on auto-start feature. If the Data Communications I ROM is given a higher priority than the FAST BASIC I ROM then the cassette power-on auto-start capability in the Data Communications I ROM will always be used instead of the one in FAST BASIC II. Therefore, it will usually be more convenient and flexible to give the FAST BASIC I ROM the high priority slot. Refer to the discussion of the relative ROM priorities in the FAST BASIC I manual under the XREF command. The flow chart below illustrates the alternative sequences possible based on the installation of Data Communications I, its relative priority to FAST BASIC I and the availability of a cassette in the internal transport.

NOTE

The power-on auto-start feature of FAST BASIC II will not operate with HP memory. This feature requires the INFOTEK EM-30 Extended Memory System in order to operate.

POWER-ON AUTO-START [Cont.]



Power-on Auto-start Control Sequence

XREF COMMAND

The XREF cross reference command prints each variable name along with the line numbers in which it appears.

SYNTAX:

XREF [# select code]

It can be executed only in the keyboard mode. The select code specification is optional and defaults to select code 15 if not used. All variables referenced in mainline memory programs are listed to the specified select code by XREF. If a Special Function key is selected only the variables referenced in the program lines on that key are listed by XREF. The variables are listed in a column according to first program reference. Each program line that the variable appears in is referenced by line number. These line numbers are listed by row next to the variable name.

The XREF command is particularly useful in large programs where it is often difficult to keep track of the variables that have been used.

Notice that function names are referenced as well as string variable names. Precision specifications of variables and arrays in COM and DIM statements are also printed. Brackets are printed after array names.

FAST BASIC II contains a complete definition of XREF exclusive of the Advanced Programming I ROM. This enables the user to obtain cross reference listings without the Advanced Programming I ROM in the machine, if desired. However, if Advanced Programming I is installed in the 9830 then refer to the discussion of relative ROM priorities in the FAST BASIC I manual under the XREF command to insure that the FAST BASIC II version is utilized. (The FAST BASIC II version ejects one additional blank line at the bottom of the listing).

The cross reference listing below results from the program listed in figure A1.

```
A[ ]   100   330   350
I[ ]   100   240   260   290   300   310   340   350
F1     120   180
S9     140   380
U1     160   210   240   250   290   330   400
U9     160   190   300   320   350   390
F9     170   340   360   360
F      180   210   260   370   370   380
W      200   230   270   270
L      330
S      350
```

LXREF COMMAND

The LXREF command is similar in function to XREF except that line numbers and their references are printed instead of variable and function names. Any line numbers in a basic program that are referenced by other basic statements will be listed along with the line numbers that refer to them. It can be executed only in the keyboard mode. The select code specification is optional and defaults to select code 15 if not used.

SYNTAX:

LXREF [# select code]

The line cross reference listing below results from the program listed in figure A1.

```
210       280

290       260     380

360       310
```

APPENDIX A
DUPLICATION OF TAPES OR DISKS

Previously, in order to duplicate tapes or disks, it was necessary to operate file by file via keyboard commands. The API ROM contains a DUP statement which allows a tape loaded in the internal cassette (SELECT CODE 10) to be duplicated on a tape loaded into a peripheral cassette (9865A). The DUP command does not allow a tape loaded in a peripheral cassette to be duplicated on the internal cassette. Moreover, the DUP command prohibits duplicating more than one source tape to a single destination tape because the command will not execute unless both tapes begin on clear leader. Using the FDATA statements, these limitations are obviated. Tapes or floppy disks may be duplicated from any select code to any other select code. Duplication can start with any source and any destination file and, in the process, various utility routines may be incorporated.

The following program duplicates tapes or disks from one select code to another:

```
100 DIM A$[100,100],I[7]
110 DISP "FIRST SOURCE FILE";
120 INPUT F1
130 DISP "NUMBER OF FILES TO COPY";
140 INPUT S9
150 DISP "FROM SELECT #, TO SELECT #";
160 INPUT U1,U9
170 F9=0
180 F=F1
190 REWIND #U9
200 W=5000
210 FIND #U1,F
220 DISP "WAITING IN CASE OF TAPE REWIND"
230 WAIT W
240 FILEID #U1,I[1]
250 BKSPACE #U1
260 IF I[1]=F THEN 290
270 W=2*W
280 GOTO 210
290 FILEID #U1,I[1]
300 MARK #U9,1,I[4]
310 IF I[3]<2 THEN 360
320 BKSPACE #U9
330 FDATA #U1,L,A$
340 I[1]=F9
350 FDATA #U9,S,I[1],A$
360 F9=F9+1
370 F=F+1
380 IF F#S9 THEN 290
390 REWIND #U9
400 REWIND #U1
410 DISP "DONE"
420 END
```

Figure A1

APPENDIX A [Cont.]
DUPLICATION OF TAPES OR DISKS [Cont.]

Line 210 finds the first source file to copy. The routine starting on line 230 tests for correct tape location. As a full cassette rewind may be necessary, the FIND statement and WAIT will recycle as many times as necessary to assure proper positioning in the worse case without undue delay for the average case. Line 300 marks a file on the destination device which is the same size as that on the source device. To avoid attempting a LOAD of an empty (type 0) or binary (type 1) file, line 310 tests the file type. If the source file is empty or binary, the program marks an equivalent file size on the destination device, increments the destination file number, and goes on to the next source file header. Otherwise, the file is loaded, the destination tape is backspaced to the beginning of the newly marked file (line 320), the data from the source is now loaded in array A at line 330, the file number in the header is changed, if necessary, to the correct file number for the destination tape (line 340) and, finally, the file data together with the new file header are stored on the destination device. Line 360 then increments the destination file number and, if the last file has not been copied, the process is repeated.

CAUTION

When duplicating files using FDATA L and FDATA S, the only parameters in the file header which may be changed are the file number, absolute file size, and data precision. If the file type is changed, any attempt to load the file may result in loss of keyboard control of the computer. In addition, it is IMPERATIVE that the last three words of the file header data be left intact when storing program files. Failure to insure that these parameters remain intact during the duplication will result in all manner of difficulties, particularly if the erroneously duplicated program is involved in a MERGE.

APPENDIX B
PACKING FILES ON TAPE

Because the 9830 cassette transports have a wide speed variation from one computer to another, the MARK statement allocates physical file lengths of a constant 1.5 times the length specified in the MARK statement. This tape is surplus to the needs of the transport which performed the MARK. However, if the tape file is rewritten by a different transport, this "slack" may be required.

The FD-30 disk speed is precise and "slack" file space is not used. All comments in this manual relative to "slack" or the "1.5 constant" apply only to cassettes, NOT the FD-30.

The 9830 writes 114 bytes of information as a file header for any type of file. Moreover, the 9830 allocates a constant 50 percent extra or "slack" tape relative to the file size specified in the MARK statement. When a MARK statement is executed, the 9830 will mark one file more than the number specified by the MARK statement. The extent to which the 50 percent excess file length may be deleted depends on certain variables. The first variable is whether or not the file will be written by a machine other than the one marking the file. If so, it will be necessary to determine experimentally the extent to which the transport speeds differ so that the file length can be made sufficiently large to accomodate the faster of the two tape transports plus some safety margin. It may be that this will not result in a significant savings. On the other hand, if the file is not to be written by a machine other than the one which performed the MARK, experiments often indicate that of all the 50 percent slack can be deleted. The slack may be eliminated entirely if the cassette is to be used in a a read only mode, regardless of machine being used.

To eliminate slack tape, all files should first be marked on the tape or disk at a size of two-thirds the number of words actually desired. FDATA S may then be used to increase the effective file size by rewriting the file headers with the "absolute file size" parameter changed to 1.5 times the marked length. This procedure may be used only when the "current file size" is set to zero. ERROR 6 results if an attempt is made to enlarge the absolute file size when the current file size is not set to zero.

CAUTION

When a tape is re-marked to pack files, 50 percent is the MAXIMUM increase in absolute file size allowable. Attempts in increase the absolute size past 50 percent will result in overwriting the following file header and consequent loss of data contained in that file. Moreover, in such a case, file numbers on the tape will become non-contiguous and normal access of files will not be possible.

This precaution also applies to tapes packed to maximum density which have been marked on one transport and subsequently are rewritten on another. If the second transport is faster, a similar loss of data will occur.

APPENDIX B [Cont.]
PACKING FILES ON TAPE [Cont.]

The following program copies a tape from one cassette transport to another, packing data and programs onto the destination tape as it duplicates. The source may be either cassette or floppy disk. The destination must be a cassette.

Lines 200 to 220 insure that both source and destination devices start on clear leader. Line 230 loads the header of the source file into array H. Line 240 calculates the ACTUAL file size required, and line 250 marks the file size. Line 270 restores the absolute file size parameter to its initial value. Lines 280 through 310 rewrite the file header to indicate the actual physical length of the file. Finally, lines 320 through 340 transfer the data from the source to the destination.

Files will become numerically non-contiguous if an existing file which is not the last file is overwrittem by a MARK statement. When a tape or disk head is positioned in an erroneous file sequence area, it will not be possible to find specified files in the customary manner.

```
100 DIM DI[100,100],HI[7]
110 DISP "SOURCE TAPE/FLOPPY SELECT CODE";
120 INPUT S1
130 DISP "DESTINATION TAPE SELECT CODE";
140 INPUT D1
150 DISP "PERCENT TAPE TO RE-MARK";
160 INPUT P
170 P=1+P/100
180 DISP "NUMBER OF FILES TO PACK";
190 INPUT N
200 REWIND #S1
210 REWIND #D1
220 IF STAT(S1)#5 OR STAT(D1)#5 THEN 200
230 FILEID #S1,H[1]
240 H[4]=H[4]/P+0.001
250 MARK #D1,1,H[4]
260 BKSPACE #D1
270 H[4]=H[4]*P
280 T=H[2]
290 H[2]=0
300 FDATA #D1,S,H[1],D
310 H[2]=T
320 FDATA #S1,L,D
330 BKSPACE #D1
340 FDATA #D1,S,H[1],D
350 N=N-1
360 IF N THEN 230
370 REWIND #D1
380 REWIND #S1
390 END
```

Figure B1

NOTE

Because of the foregoing, the **re**-marking of an existing file which is not the last file should be avoided. When it cannot be avoided, it must be understood that the balance of all files may very well have to be rewritten in order to maintain contiguous file sequence.

In order to prevent loss of information, specific precautions must be taken with respect to duplicating all of the files which will be affected by the new MARK statement. The number of files that will be affected can be computed, and appropriate precautions taken to **avoid loss of data.**

One final note regarding the marking of files relates to passing of system control during a FIND command. If a FIND command is executed, and tape motion stop is **not confirmed** prior to executing a MARK command, a file other than that specified in the FIND command **may** be overwritten and data contained in that file will be lost.

APPENDIX C
SECURING FILE INFORMATION

The FDATA load and FDATA store statements permit loading any type of file except binary into a specified array. The EXOR function of FAST BASIC I can then be used to exclusively OR each element of the array against a key number, thereby essentially scrambling its content. The array thus coded can be stored back in its original file on tape or disk.

When a number is exclusively OR'ed to a given key number and the result exclusively OR'ed to the same key number, the original number will be obtained. To de-secure a file, therefore, the process is repeated using the same key number.

The process results in extreme security. Over 65,000 keys are possible as any number between -32,768 and 32,767, except for Ø, is a valid key. Moreover, each time an attempt is made at operating a program when an incorrect key is used, keyboard control of the machine will be lost, forcing machine power to be turned off and back on again to regain control.

Remember that the contents of a program file can be examined by listing an array. As the first word is the first program line number, it is a simple matter to compute the key which will yield a 10 when exclusively OR'ed to this first number.

To gain maximum security, the first program line number should be arbitrary. The balance of the line numbers are of no consequence as their location in the array (after the exclusive OR process) is no longer evident.

A further means of completely confounding an attempt at de-securing a program file is to use an array of random numbers which contains, in part, the program to be secured. SEND, of course, is a straightforward means of making the program file a part of the random number array. In this case, the first element of the random number array may contain a 10. If this is done, the current file length parameter must be made to conform to the number of array elements used.

The following program is a straightforward secure and de-secure routine which operates by loading the specified file into array P. The nested FOR-NEXT loops beginning on line 140 exclusively OR each element of the array with the key number, KØ. The SCAN statement in the line 110 sets LØ equal to the location of the first undefined element in the program array. The FOR-NEXT loop decrements this number and terminates the exclusive OR operation when the number equals Ø, thereby avoiding an ERROR 40. At line 210, the tape or disk is backspaced to the beginning of the file and line 220 stores the array.

APPENDIX C [Cont.]
SECURING FILE INFORMATION [Cont.]

```
10 DIM P[100,100],F[7]
20 DISP "FILE NO. TO SECURE OR DE-SECURE";
30 INPUT F0
40 DISP "SELECT CODE";
50 INPUT S0
60 FIND #S0,F0
70 WAIT 5E+03
80 FILEID #S0,F[1]
90 IF F[1]#F0 THEN 60
100 FDATA #S0,L,P
110 SCAN P=-32768,L0
120 DISP "SECURE KEY";
130 INPUT K0
140 FOR R=1 TO 100
150 FOR C=1 TO 100
160 L0=L0-1
170 IF  NOT L0 THEN 210
180 P[R,C]=EXOR(P[R,C],K0)
190 NEXT C
200 NEXT R
210 BKSPACE #S0
220 FDATA #S0,S,F[1],P
230 DISP "DONE"
240 END
```

Figure C1

INFOTEK FAST BASIC II

INSTRUCTION MANUAL

Infotek Systems

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

5007611 PRINTED IN U.S.A.