# BASIC COMPILER


FEBRUARY 1980



INFOTEK SYSTEMS
1400 North Baxter Street
Anaheim, California 92806

TABLE OF CONTENTS


| | PAGE |
|---|---|
| Introduction | 1-2 |
| What To Compile | 3 |
| Table I and II | 4 |
| Table III and IV | 5 |
| Statements and Commands | 6 |
| Value Statement | 7 |
| Arry Statement | 8 |
| Integer Statement | 9 |
| EDEF Statement | 10 |
| FCOMP Command | 11-12 |
| BLIST Command | 13 |
| Examples | 14-15-16 |
| Summary of Errors | 17 |
| Table V | 18 |
| Appendix A | 19 |
| Appendix B | 20 |

# INSTRUCTION MANUAL
# FOR THE INFOTEK SYSTEMS
# BASIC COMPILER

## INTRODUCTION

The Infotek BASIC Language Compiler is a new step forward for the HP 9830 user. The compiler allows the user to create new statements by compiling BASIC language programs into a binary form. The new statements perform the same function as the programs from which they were compiled and can be used just as if they were in a ROM, such as String Variables or Matrix Operations. The advantages of this capability are two-fold: the user can now select portions of his programs which are slow due to a high instance of numerical calculations or repetitive procedures and increase their speed significantly. In addition, frequently used routines can be compiled and saved in a library-like binary for run-time loading, saving the user the time and unnecessary work of repeated linking of useful routines.

Program execution via compilation is different from that by interpretation, to which the 9830 user is accustomed. The "operating system" of the 9830, which is written in assembly language, interprets a BASIC language statement in the following manner: it encounters the symbolic representation of a BASIC language statement in R/W memory, searches for the assembly language routine to execute it by checking binaries, option ROMs and BASIC ROM, and then either executes it by branching control to the proper routine or issues an error message to indicate that no appropriate routine can be found. The process of accessing a variable requires a similar search through R/W memory.

The implications of this method of executing a program are (1) that program lines are easy to change since the interpreter does the work of locating the appropriate code and variables each time a program line is encountered, (2) programs can be stored very efficiently in symbolic representation, and (3) programs run with the time-consuming overhead of the interpretation for each statement. By contrast, a compiled program has no overhead of interpretation. A compiled program is generated by a program called a compiler, which translates the BASIC language statements and variable references into assembly language instructions "in line", and saves the results for later execution. This eliminates the need for an additional program (the interpreter) to locate the needed routines, as they are all present in a "line" through which the program flows. As an example, in a compiled program the code required to execute a LET would appear as many times as a LET appeared. Any variable references in the LET statement would be resolved (which is to say, the variable located in the machine's memory) at compile time, avoiding the variable search at run time. The implications of the compiled version are (1) a speed increase

due to the elimination of the interpretive overhead, but (2) larger program storage requirements and (3) loss of the ability to easily change programs as the program must be re-compiled each time it is changed.

In the case of the 9830, the benefit of compilation is diminished somewhat by the fact that the interpretive environment still exists. Specifically, let us assume we have a subroutine which calculates the Fourier transform of a waveform stored in discrete form in an array. By compiling this subroutine, we gain a dramatic speed increase due to the non-interpretive execution of the numerically complicated and highly repetitive Fourier transform code, but the overhead mentioned before still exists in recognizing any references to the compiled version and locating it each time it is referenced. In short, the user can use the compiler to optimize his programs by introducing new statements and using them just as if they were in a ROM.

The Compiler is a two ROM set located on your MX-30 BASIC ROM Board. This and the Compiler Execution ROM are required to compile BASIC language programs and save the resultant binary. However, only the Execution ROM is required to properly execute any compiler-generated binary program. The Execution ROM works with any 9830 configuration (stock or FP-30) which allows binaries created on one machine to be transferred to another via cassette, floppy, or hard disk and loaded via the LOADBIN or GETB statements.

# HP Computer Museum
# www.hpmuseum.net

**For research and education purposes only.**

## WHAT TO COMPILE?

The question of which programs to compile is best answered by examining the capabilities of the compiler itself. For example, the compiler does not support string operations, as no appreciable advantage can be achieved over the String Variables ROM itself. From the tables of supported statements (Tables I and II) and predefined functions (Table III), one can see the orientation is primarily numerical, and then syntactical. Consequentially the compiler is best used to speed the execution of numerical routines, and routines which are executed repeatedly. However, the execution speed of any routine with a significant number of variable references will be enhanced by compilation due to the elimination of the time consuming variable searches. Most routines realize a threefold increase in execution speed from compilation, but routines with large numbers of variable references are likely to execute even faster.

The strategy to follow in implementing compiled programs depends on whether or not the target program is already written. If the program is already written and it would be beneficial to compile a part of it, the problem of statements that are not supported by the compiler, yet appear in the part in question, may arise. One answer would be to rewrite or eliminate the unsupported statement. Another is to compile the sections around the unsupported statement.

Authors of new programs may take full advantage of the compiler by planning for routines which are numerical, or repetitive, or make use of many variables to be compiled. These new programs can then be organized in such a fashion to avoid the use of unsupported statements.

In developing new routines, the fact that minimal error checking has been included in the Compiler and Execution ROMs (to maximize speed) should be remembered. It is advised that the new routine be debugged interpretively prior to compilation.

BEEP
CFLAG
DEG
DEF
DISP
END
FOR-NEXT†
GOSUB AND GOSUB-OF
GOTO AND GOTO-OF
GRAD
IF-THEN†
LET (AND IMPLIED LET, I.E. A=B)
PRINT
RAD
REDIM
REM
RETURN
SFLAG
STOP
WAIT
WRITE(S,*) ***NOTE: FORMATTED WRITES NOT SUPPORTED***

TABLE I
STATEMENTS RECOGNIZED BY THE COMPILER AND EXECUTION ROM FOR ANY 9830
†See Appendix A

ESEND
HUNT
MAT A=
SORT

TABLE II
ADDITIONAL STATEMENTS SUPPORTED BY THE COMPILER AND EXECUTION ROMS FOR MX-30 MACHINES ONLY (ERROR 2000 RESULTS IF THESE STATEMENTS ARE EXECUTED ON A NON-MX-30 MACHINE)

| | |
|---|---|
| ABS | RBYTE |
| ACS | REC |
| ASN | RND |
| ATN | ROT |
| BIAND | ROW |
| CMP | RVI |
| COL | RWORDS |
| COS | SGN |
| DEC | SHF |
| DET | SIN |
| EXOR | SPA |
| EXP | SQR |
| FLAG | SUNIT |
| FSIZE | TAB |
| INT | TAN |
| KEY | TEOT |
| LGT | TRIG |
| LIN | TYP |
| LOG | USTAT |
| MOD | WBYTE |
| OCT | |

TABLE III
PREDEFINED FUNCTIONS RECOGNIZED BY THE COMPILER AND EXECUTION ROMS

FC
LEN
NUM
POS
RES
UND
VAL
VCOL
VROW

TABLE IV
PREDEFINED FUNCTIONS NOT RECOGNIZED BY COMPILER AND EXECUTION ROMS

## STATEMENTS AND COMMANDS IN THE COMPILER AND EXECUTION ROMS

NOTE: In the following descriptions [...] means the item in brackets is optional, and may be omitted.

### THE DFSTAT STATEMENT

The DFSTAT statement must be the first statement in the program to be compiled. It tells the compiler what the name of the statement is, and what the names of parameters to it are. These represent values or variables which will be passed to the compiled program when called at run-time. These are much like the argument to a user defined function, with the exception that more than one parameter may be defined.

All other variables are considered "local", and will not be accessible outside the compiled program at run-time. It is important to note that access to local variables is substantially faster than access to parameters.

NOTE: The name must be specified exactly as desired. In particular, spaces are ignored by the 9830 and so should not be included in the name. The compiler allows 20 names to be included in one binary with an average of 5 characters each. It is, however, possible to have less names with more characters per name.

SYNTAX:

```
DFSTAT "<statement name>",<variable name> [...,<variable
       name>]
```

EXAMPLE:

```
DFSTAT "FACTORIAL",A,B,
```

The example defines a statement named FACTORIAL, with parameters A and B.

## THE VALUE STATEMENT

The VALUE statement defines one or more parameters to be passed "by value", which means a value will be assigned to the parameter when the statement is called. However, after that it will be treated as a local variable and referenced substantially faster. This means that a variable passed as a value parameter may not be modified. It also allows an expression to be passed to the compiled program, whereas a normal parameter, being passed by reference, can be changed and must be a variable only.

An error 1003 will result if the variable is not a parameter (declared in the DFSTAT statement) or if it has been declared as an array. All VALUE statements must appear between the DFSTAT statement and the first executable statement of the sub-program.

SYNTAX:

VALUE <variable name> [...,<variable name>]

EXAMPLE:

```
VALUE B
```

## THE ARRY STATEMENT

The ARRY statement declares a parameter to be an array. If it is not so declared, errors will result if the parameter is used in the compiled program as an array. ARRY parameters are "called by name", which means any modification of an array which occurs in a compiled program will also affect the array which was passed.

Error 1003 will result if the variable was not declared as a parameter (in the DFSTAT statement) or if it was previously declared as a value parameter. All ARRY statements must occur between the DFSTAT statement and the first executable statement of the compiled program.

SYNTAX:

```
ARRY <array variable name> [...,<array variable name>]
```

EXAMPLE:

```
ARRY A
```

Defines A to be an array.

Note that A may still be used as a local simple variable, i.e.

```
10  DFSTAT "NAME",A
20  ARRY A
30  A=0
40  A(1)=A
```

## THE INTEGER STATEMENT

The INTEGER statement allows the declaration of a local variable as an integer. Its sole purpose is to speed array subscripting, which it does by about 10%.

SYNTAX:

INTEGER <local variable name> [..., <local variable name>]

EXAMPLE

```
10  DFSTAT "SUMSQUARES" A,B
20  ARRAY A
30  INTEGER I
40  FOR I= 1 TO ROW(A)
50  B=B+A(I)*A(I)
60  NEXT I
70  END
```

is approximately 10% faster than it would be if line 30 were omitted.

## THE EDEF STATEMENT

The EDEF statement is for the use of the compiler only. It has no execution properties whatsoever. It is used to define the end of a multi-line user defined function. It is used for multiline functions only. It must be placed after the last return statement in the function.

SYNTAX:

```
EDEF
```

EXAMPLE:

```
10  DEF FNA(X)
20  IF A <0 THEN 40
30  RETURN 0
40  RETURN 1
50  EDEF
```

This function returns 1 if A is negative, 0 else. The EDEF tells the compiler that the function is done.

## THE FCOMP COMMAND

The FCOMP command tells the compiler to compile the BASIC program currently in memory, and optionally to store it on floppy/cassette. To compile a BASIC program, follow the following steps:

1. Execute a SCRATCHALL to delete any binary currently in memory.

2. Load the program you wish to compile into the 9830's memory. If necessary, add the DFSTAT and any needed VALUE, ARRY, or INTEGER statements. The statements in the subprogram must consist only of the statements listed in Tables I, II and functions in Table III.

3. Execute the FCOMP command. If storing to floppy/cassette, be sure the drive is ready and the diskette/tape installed.

Upon completion of the compilation, a binary containing the statement just defined will reside in main memory, provided no errors are detected. (See Appendix B for details about memory usage by the binary). In addition, if the optional parameters to the FCOMP command were given, the binary has been stored on floppy/tape. If an error is detected, no binary is created and nothing will be stored. In fact, if linking in a new statement (see below) the old binary will be scratched from memory. To store the binary on a 9880A/B Mass Memory, use the SAVEB statement of the Infotek Mass Memory II ROM.

To link a new statement into an existing binary (up to a maximum of 20 statements per binary), substitute the following for step 1 above:

1. If the binary you wish to link to is in memory (possibly from a previous compilation), proceed. Otherwise load it in by using the LOADBIN or GETB statements.

Then execute steps 2 and 3 as above. The new statement name will be added to the binary in memory, and stored on diskette/tape if requested. It is only possible to link to binary programs created by the compiler.

**SYNTAX:**

FCOMP [ <select code> [, <FILE NO.>] ]

EXAMPLE:

FCOMP
Compiles the program currently in memory and creates a binary in main memory.

FCOMP 5,3
Compiles the program currently in memory and creates a binary in main memory and on select code 5, file 3.

## BLIST COMMAND

The BLIST command lists the names of the statements in the binary currently in main memory. This command is in the Execution ROM and does not require the Compiler ROM. This command is very important: since all binary programs generated by the compiler have the same system identification code, ERROR 1 will not be generated if the wrong binary is loaded by mistake. However, the program will not run correctly if the wrong binary is loaded.

EXAMPLE USAGE

Examine the following sample program:

```
10  DISP "INPUT N";   *
20  INPUT N
30  R=1               *
40  FOR I=2 TO N      *
50  R=R*I             *
60  NEXT I            *
70  PRINT R           *
80  GOTO 10           *
```

The statements marked with a * are supported by the Compiler and Execution ROMs. There is no benefical reason to compile the DISP or PRINT statements in this case, but the actual computation of the factorial could be made into a statement by compiling the following:

```
10  DFSTAT "FACTORIAL",A,B
20  VALUE A
30  B=1
40  FOR I=2 TO A
50  B=B*I
60  NEXT I
70  END
```

When compiled, this will define a statement to compute factorials, given two inputs: The value for which to compute the factorial (A) and the result variable (B). This statement will be called FACTORIAL. With the resultant binary in memory, the original program can be written to look like this:

```
10  DISP "INPUT N";
20  INPUT N
30  FACTORIAL N,R
40  PRINT R
50  GOTO 10
```

Note that the variable used in the call as the return variable does not have to be the same as that used in the DFSTAT statement. Also notice that because the first parameter (A) was declared as a value parameter, an expression (7) may be used instead of a variable name.

EXAMPLE:

```
FACTORIAL 7,Z
```

Would set Z to 5040 (7!). However,

```
10 FACTORIAL Z1,7
```

would result in an error 6, since the second parameter cannot be an expression. Also note that the value of all variables will remain unchanged, except that used as the second parameter (Z in the last case).

The following demonstrates array usage in a compiled program:

```
10  MATREAD #1;A
20  X0=N1=0          *
30  N=ROW(A)         *
40  FOR J=1 TO N     *
50  FOR K=1 TO N     *
60  X0=X0+ABS(A[J,K])*
70  N1=N1+1          *
80  NEXT K           *
90  NEXT J           *
100 X0=X0/N1         *
110 PRINT "AVERAGE ELEMENT =",X0 *
120 END              *
```

Again, all the statements marked with an asterisk are supported by the Compiler and Execution ROMs but, as before, the PRINT statement in the compiled program might limit its usefulness as a general purpose routine. Suppose we compile the following:

```
10  DFSTAT "AVG",A,B1
20  ARRY A
30  X0=N1=0
40  N=ROW(A)
50  FOR J=1 TO N
60  FOR K=1 TO N
70  X0=X0=ABS(A[J,K])
80  N1=N1+1
90  NEXT K
100 NEXT J
110 B1=X0/N1
120 END
```

The original program can now be written as follows:

```
10  MATREAD #1;A
20  AVG A,X0
30  PRINT "AVERAGE ELEMENT=",X0
40  END
```

The routine can now be saved and used any time the average of a square matrix is desired (the reason we left the PRINT statement out). With small modifications, different sorts of averages could be computed by different routines, or by a single general purpose routine with a parameter indicating which to do.

The DFSTAT, VALUE, ARRY, EDEF, and INTEGER statements are used by the compiler only. They have no effect on the execution of a program, and therefore can be left in while debugging the program interpretively without any effect.

:

SUMMARY OF ERRORS:

| | |
|---|---|
| 1: | If ERROR 1 while loading binary, then the Execution ROM is not installed. A SCRATCHALL should be executed immediately, since the Execution ROM is required for proper functioning of the compiler generated binary. |
| 999: | The compiler has tried to do something horrible and caught itself. Contact Infotek. |
| 1000: | DFSTAT is missing or not the first statement in the subprogram. |
| 1001: | One of the compiler ROM set is missing, probably the Execution ROM. |
| 1002: | Insufficient memory to complete compilation. |
| 1003: | Parameter error, variable not declared as parameter, variable declared twice, array variable declared as value, or vice-versa. |
| 1004: | No room for statement name or attempt to put more than 20 statements into one binary. |
| 1005: | Statement not recognized by compiler. |
| 1006: | Function not recognized by compiler. |
| 1027: | FOR variable must be local variable, cannot be parameter. |
| 1032: | Formatted WRITE statement not recognized. |
| 1041: | Undeclared array. |
| 1042: | Constant subscript out of range. |
| 1044: | Jump to undefined line, or no end statement. |
| 1047: | Improper return statement. |
| 1048: | Improper FOR-NEXT matching. |
| 2000: | Attempt to use MX-only binary on non-MX machine. |
| 2001: | Option ROM not found. Type ROM?<EXECUTE> to find out which one. See Table V to determine which ROM from number displayed. |
| 2048: | Improper FOR loop nesting. |

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

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

TABLE V

ROM NAME TO ROM NUMBER CORRELATION

## Appendix A

Due to the internal implementation of the FOR-NEXT and IF-THEN statements, a speed gain of about 5% can be realized by replacing FOR-NEXT loops with IF-THEN loops in compiled programs as follows:

```
.
.
.
1000 I=1
1010 ...
.
.    Body of Loop
.
2000 I=I+1
2010 IF I<=N THEN 1010
.
.
.
instead of
.
.
.
1010 FOR I=1 TO N
.
.
.
2010 NEXT I
```

Appendix B

Binary programs created by the compiler appropriate R/W memory in 1K blocks. The following formula may be used to estimate the amount of memory required for a compiler-generated binary (in words):

C=2*B+150

When C is rounded up to the nearest 1K, B represents the length of the BASIC language program to be compiled in words. This number may be found by subtracting the remaining memory shown in the display after a LIST command (with the program in memory but no variables initialized from the memory shown when a LIST is executed at power up. The constant of 150 is for compiler overhead.

For example:

If B=1250, C=2*1250+150

=2650

So, the binary would take 3144 words for 3K.

If B=436 C=2*436+150

=1022

rounded up, C would be 1024 or 1K, but since the formula is only an approximation, the binary could easily take 2K depending on the specific program.