tiny BASIC for the F8

BY JERRY D. FOX
3 High Ridge Road
Granby, CT 06035

Herewith is documentation and the source for a tiny BASIC Interpreter written for the F8 microcomputer. The structure of the program is copied from Palo Alto tiny BASIC written by Dr. Li-Chen Wang. It is to Dr. Wang's credit that he wrote such an elegant program that it was fairly easy to convert to F8 from 8080 assembly language.

The program contains software stack so the nesting of GOSUB's and FOR-NEXT loops can be accommodated. All I/O uses a monitor called FAIRBUG written by Fairchild. -Jerry D. Fox

TINY BASIC FOR THE F8

Numbers. All numbers are integers and must be less than 32767.

Variables. There are 26 variables noted by letters A through Z. There is also a single array @ (I). The single array @ (I) is equivalenced to A-Z in the following manner. @(−I) = A, @(−26) = Z. In addition to A-Z and @(I) being variables, they do double duty as string variables. A$-Z$, @$(I) refer to the same physical locations but permit string operations.

Functions. There are 4 functions:

ABS(X) gives the absolute value of X.

RND(X) gives a random number between 1 and X (inclusive).

PEEK(X) retrieves the byte at addr X.

LEN(X) if X=0, gives the length of the last referenced string if X = @$(I), or A$ to Z$, it gives the length of the referenced string.

Arithmetic and Compare Operators.

/	divide.
*	multiply.
−	subtract.
+	add.
>	greater than (compare).
<	less than (compare).
=	equal to (compare).
#	not equal to (compare).
>=	greater than or equal to (compare).
<=	less than or equal to (compare).

+, −, *, and / operations result in a value between −32767 and 32767. (−32768 is also allowed in some cases.) All compare operators result in a 1 if true and a 0 if not true.

Expressions. Expressions are formed with numbers, variables, and functions with arithmetic and compare operators between them. + and − signs can also be used at the beginning of an expression. The value of an expression is evaluated from left to right, except that * and / are always done first, and then + and −, and then compare operators. Parentheses can also be used to alter the order of evaluation. Note that compare operators can be used in any expression. For example:

```
10 LET A= (X > Y) * 123 + (X=Y) * 456 + (X > Y) * 789
20 IF (U=1) * (V<2) + (U>V) * (U<99) * (V>3) PRINT
                                              "YES"
30 LET R= RND(100), A = (R>3) + (R>15) + (R>56) +
                                              (R>98)
```

In statement 10, A will be set to 123 if X > Y, to 456 if X = Y, and to 789 if X < Y. In statement 20, the "*" operator acts like a logical AND, and the "+" operator acts like a logical OR. In statement 30, Y will be a random number between 0 and 4 with a prescribed probability distribution of: 3% of being 0, 15−3=12% of being 1, 56−15=41% of being 2, 98−56=42% of being 3, and 100−98=2% of being 4.

Direct Commands. All the commands described later, except strings, can be used as direct commands except the following three, they can only be used as direct command and not as part of a statement:

RUN will start to execute the program starting at the lowest statement number.

LIST will print out all the statements in numerical order.

LIST 120 will print out all the statements in numerical order starting at statement 120.

LIST 120, N will print N lines (up to a maximum of 255) starting at 120.

NEW will delete all statements.

Abbreviation and Blanks. You may use blanks freely, except that numbers, command key words, and function names cannot have embedded blanks. You may truncate all command keywords and function names and follow them by a period. "P.", "PR.", "PRI.", and "PRIN." all stand for "PRINT". Also the word LET in LET command can be omitted. The "shortest" abbreviation for all keywords are as follows:

A. = ABS	D. = DATA	C. = CALL	F. = FOR	GOS. = GOSUB
G. = GOTO	IF = IF	IN. = INPUT	L. = LIST	LEN = LEN
M. = MON	N. = NEW	N. = NEXT	P. = PRINT	PE. = PEEK
PO. = POKE	REM = REMARK	R. = RETURN	R. = RND	R. = RUN
RES. = RESTORE	READ = READ	S. = STEP	S. = STOP	TO = TO
T. = THEN	IMPLIED = LET			

Statements. A statement consists of a statement number of between 1 and 32767 followed by one or more commands. Commands in the same statement are separated by a semi-colon ";". "GOTO", "STOP", and "RETURN" commands must be the last command in any given statement.

Commands. tiny BASIC commands are listed below with examples. Remember that commands can be concatenated with semi-colons. In order to store the statement, you must also have a statement number in front of the commands. The statement number and the concatenation are not shown in the examples.

- REMor REMARK Command

  REM anything goes

This line will be ignored by TBI.

- LET Command

  LET A = 234−5*6, A = A/2, X = A−100, @(X+9) = A−1

will set the variable A to the value of the expression 234−5*6 (i.e., 204), set the variable A(again) to the value of the expression A/2 (i.e., 102), set the variable X to the value of the expression A−100 (i.e., 2), and then set the variable @(11) to 101 (where 11 is the value of the expression X+9 and 101 is the value of the expression A−1).

  LET U = A # B, V = (A > B) * X + (A < B) * Y

will set the variable U to either 1 or 0 depending on whether A is not equal to or is equal to B; and set the variable V to either X, Y or 0 depending on whether A is greater than, less than, or equal to B.

- PRINT Command

  PRINT will cause a carriage-return (CR) and a line-feed (LF) on the output device.

  PRINT A * 3 + 1, "ABC 123 !@#", ' CBA ' will print the value of the expression A * 3 + 1 (i.e., 307), the string of characters "ABC 123 !@#", and the string "CBA", and then a CR-LF. Note that etiher single or double quotes can be used to quote strings, but pairs must be matched.

  PRINT A*3+1, "ABC 123 !@#", ' CBA ',

will produce the same output as before, except that there is no CR-LF after the last item printed. This enables the program to continue printing on the same line with another "PRINT".

  PRINT A, B, #3, C, D, E, #10, F, G

will print the values of A and B in 6 spaces, the values of C, D, and E in 3 spaces, and the values of F and G in 10 spaces. If there are not enough spaces specified for a given value to be printed, the value will be printed with enough spaces anyway.

  PRINT 'ABC' , – , 'XXX'

will print the string "ABC", a CR without a LF, and then the string "XXX" is overprinted on "ABC" followed by a CR-LF.

- INPUT Command

  INPUT A, B

When this command is executed, tiny BASIC will print "A:" and wait to read in an expression from the input device. The variable A will be set to the value of this expression. Then "B:" is printed and variable B is set to the value of the next expression read from the input device. Note that not only numbers, but also expressions can be read as input.

  INPUT 'WHAT IS THE WEIGHT'A, "AND SIZE"B

This is the same as the command above, except the prompt "A:" is replaced by "WHAT IS THE WEIGHT:" and the prompt "B:" is replaced by "AND SIZE:". Again, both single and double quotes can be used as long as they are matched.

  INPUT A, 'STRING', – , "ANOTHER STRING", B

The strings and the "–" have the same effect as in "PRINT".

- IF Command

  IF A$ = 'NO' THEN @$(I) = 'RIGHT'

  IF A < B LET X=3; PRINT 'THIS STRING'

will test the value of the expression A < B. If it is not zero (i.e., if it is true), the commands in the rest of this statement will be executed. If the value of the expression is zero (i.e., if it is not true), the rest of this statement will be skipped over and execution continues at next statement. Note that the word "THEN" is optional. It may or may not be used.

- GOTO Command

  GOTO 120

will cause the execution to jump to statement 120. Note that GOTO command cannot be followed by a semi-colon and other commands. It must be ended with a CR.

  GOTO A*10+B

will cause the execution to jump to a different statement number as computed from the value of the expression.

- GOSUB and RETURN Commands

GOSUB command is similar to GOTO command except that: a) the current statement number and position within the statement is remembered; and b) a semi-colon and other commands can follow it in the same statement.

  GOSUB 120

will cause the execution to jump to statement 120.

  GOSUB A*10+B

will cause the execution to jump to different statements as computed from the value of the expression A*10+B.

- RETURN. A RETURN command must be the last command in a statement and followed by a CR. When a RETURN command is encountered, it will cause the execution to jump back to the command following the most recent GOSUB command.

GOSUB can be nested. The depth of nesting is limited only by the stack space.

- FOR and NEXT Commands

  FOR X=A+1 TO 3*B STEP C−1. The variable X is set to the value of the expression A+1. The values of the expressions (not the expressions themselves) 3*B and C−1 are remembered. The name of the variable X, the statement number and the position of this command within the statement are also remembered. Execution then continues the normal way until a NEXT command is encountered. The STEP can be positive, negative or even zero. The word STEP and the expression following it can be omitted if the desired STEP is +1.

  NEXT X. The name of the variable (X) is checked with that of the most recent FOR command. If they do not agree, that FOR is terminated and the next recent FOR is checked, etc. When a match is found, this variable will be set to its current value plus the value of the STEP expression saved by the FOR command. The updated value is then compared with the value of the TO expression also saved by the FOR command. If this is within the limit, execution will jump back to the command following the FOR command. If this is outside the limit, execution continues following the NEXT command itself.

FOR can be nested.The depth of nesting is limited only by the stack space. If a new FOR command with the same control variable as that of an old FOR command is encountered, the old FOR will be terminated automatically.

- CALL Command: Call (X) calls the routine at address X.

- POKE Command: POKE X,Y at address X place the 1 byte value represented by Y.

- STRINGS

```
A$ = "ABC$1234XYZ"
X = LEN (0)
INPUT  @$(I)
IF A$ = @$(I)  GOTO 100
M$ = A$
```

String variables are physically the same as simple variables but logically different. In the above examples the physical locations A will contain AB, B will contain C$, and F will contain ZZ (odd length repeats the last character for compare purposes). X would be set to 11, the length of the last referenced string.

The maximum string length is 72 characters. The compare of A$ and @$(I) is not a full string length compare, only the single physical byte A and @(I) are compared. By using the LEN function FOR-NEXT and the fact that @(–1)=A one can easily compare a whole string. It is obvious that the user must be careful or strings could smash variables that could cause errors. Strings cannot be executed in the direct mode.

- READ and DATA Commands

```
100 DATA 1,0,  'YES', – 10
200 DATA (A + B) * C,  1000/33,  'HELP'
300 READ I,  B,  C$
400 READ E,  F,  @(0),  @$(I)
```

A read statement is used to assign to the listed variables values obtained from a DATA statement. The variables are read in a one-to-one correspondence. In the example: I will = 1 and @$(I) will equal HELP.

Each time a READ statement is encountered in the program, the READ resumes at the location in the DATA statements where reading previously stopped. Location of DATA statements are unimportant, but they must be in the correct sequential order.

- RESTORE Command: The RESTORE command returns the READ-DATA pointer to its original position so data may be reread.

- MON Command: A user option to return control to a monitor.

- STOP Command: This command stops the execution of the program and returns control to direct commands from the input device. It can appear many times in a program but must be the last command in any given statement, i.e., it cannot be followed by a semi-colon and other commands.

Error Report. There are only three error conditions in tiny BASIC. The statement with the error is printed out with a question mark inserted at the point where the error is detected.

1. WHAT? means it does not understand you. Example:
   WHAT?
   210 P?TINT "THIS" where PRINT is mistyped
   WHAT?
   260 LET A = B + 3, C = (3 + 4?, X = 4
2. HOW? means it understands you but does not know how to do it.
   HOW?
   310 LET A = B*C? + 2 where B*C is greater than 32767
   HOW?
   380 GOTO 412? where 412 does not exist
3. SORRY means it understands you and knows how to do it, but there is not enough memory to do it.

Error Corrections. If you notice an error in typing before you hit the CR, you can delete the last character by the Rub-Out key or delete the entire line by the Alt-Mode key. tiny BASIC will echo the deleted character for each Rub-Out. Echo for Alt-Mode consists of a LF, a CR, and an up-arrow.

To correct a statement, you can retype the statement number and the correct commands. tiny BASIC will replace the old statement with the new one.

To delete a statement, type the statement number and a CR only.

Verify the corrections by "LIST nnnn, 1".

Symbol Table

```
STAT  0003   ST1   0009   ST2   000B   ST3   000D   FLIE  005C
INST  0067   NLIF  0075   ST4   007F   ST5   0087   TxCK  008F
TxC1  00A6   GETN  00A9   GET1  00E1   GET2  00B4   GET3  00C1
GET4  00CC   GET5  00D7   FNDN  00ED   FNDL  00F5   FND1  00F1
FND2  00FD   FND3  00FE   FNDT  010F   FND4  0110   PRTN  0117
PRTG  0130   PRT1  0134   PRT2  0142   PRTM  0145   XCV1  0156
XCV2  0169   XCV3  0171   XCV4  0174   XCV5  017D   XCV6  0183
PBLK  0186   QST6  018C   QST1  0195   QST2  019A   QST3  01A5
QST4  01AA   QST5  01AE   QST6  01B8   TESL  01BD   TVS   01DB
TVE   01F5   TVT   01FF   QSCY  0211   ASOY  0214   TV0   021A
TV1   0223   TV2   0225   TV3   022A   TVR   022E   TSTM  022F
TS1   0236   TS2   0241   TS3   0242   QHOW  025B   AHOW  025E
DIRT  0269   EXEC  0267   EX1   026C   EX2   027E   EX3   0285
EX4   0289   INPR  0290   INPT  029F   IP1   02A2   IP2   02B1
IP3   02D0   IPX   02FA   IPS   0304   IP4   030C   FINI  0316
FIN   0319   DFFT  031F   LET   032A   PRIT  0335   FR1   0344
PR2   034E   PR3   035B   PR4   0363   PR5   036E   PR6   0374
PR7   0389   PR8   038C   GOSB  0391   GOS1  03A2   RETN  03B4
RET1  03C1   WHAT  03D6   HOW   03DB   SORY  03DF   LIST  0400
LIS1  041A   LIS2  041E   LIS3  0421   LIS4  042C   LIS5  042D
LIS6  0436   LIS7  0445   NEW   044B   NEWT  044E   STOP  045B
IF    045E   REM   0469   IF1   0471   FOR   0474   FR1   048F
FR2   049A   FR3   049F   FR4   04A3   FR5   04B5   FR6   04BF
FR8   04D9   NEXT  04EE   NX0   04EE   NX1   04F8   NX2   050D
NX3   053D   NX4   0540   NX5   0553   NX6   055B   READ  055C
REA1  0567   REA2  0580   DATA  059B   DAT1  05A6   RESE  05AD
GOTO  05B5   GO1   05CB   RUN   05CB   RNXL  05D1   RTSL  05DD
RSML  05E7   FINH  05F0   EXPR  05FD   EXP1  0607   XP11  060F
XP12  0615   XP13  061C   XP14  0625   XP15  062E   XP16  0635
FALE  063A   TRUE  063B   XP17  063E   XP1R  0644   XP19  0660
EXP2  0669   XP21  067A   XP22  0681   XP23  0684   XP24  0693
XP25  06A9   XP26  06BD   EXP3  06BE   XP33  06C5   XP31  06C6
XP32  06EB   XP33  06F9   XP34  06FC   XP35  0729   EXP4  073A
XP40  0745   XP41  0763   XP42  0775   XP43  07B6   XP44  07D1
XP45  0796   PARN  0799   PAR1  07D0   GWHT  07H2   ALHT  07E5
ERRR  07B8   ENDR  07F3   RDAA  07FC   SETL  0802   SETH  0815
SET1  0830   SET2  0834   SET3  0841   SET4  0844   SETR  0849
CHKN  084C   CHGN  0852   CHG1  085E   CHG2  0862   SUBD  0863
NCS   0871   MTPY  0875   ADDD  0879   MULT  087B   MUL1  08B6
DIVE  08BB   DV1   08A5   DV2   08A9   DV3   08AC   COMP  08BD
COMX  08BE   COMT  08BF   COM1  08C6   SKIP  08CE   CHAR  08CF
IGNK  08D8   IGN1  08D9   EXCH  08E0   MV21  08F1   MVUP  08FA
MV1   0904   MV2   0908   MVDN  090C   MVD1  0914   BUFV  0922
BU0   092E   BU1   0938   BU2   0946   BU5   095F   BU6   0962
BU4   096E   BU7   0971   BU8   0980   SLEN  0984   SAVE  098E
SAV1  09A1   SAV2  09B1   SAV3  09BB   RESR  09BE   REFR  09C9
RES1  09CD   RES2  09D2   PUSC  09D5   PULC  09E1   PUS0  09E9
PUSR  09EA   PUL0  09F6   PULR  09F7   PUST  0A00   PULT  0A0C
POPT  0A15   USR   0A1E   POKE  0A2F   POK1  0A4A   APOT  0A4D
QUOE  0A51   AQ1   0A53   AQ2   0A5B   AQ3   0A66   AQ4   0A75
LEN   0AB2   ABS   0A92   RND   0AA0   RN1   0AC4   PEEK  0AE4
TAB   0AF5   TA1   0AFF   UMFL  0B0F   CHR   0B3B   CH1   0B10
HB00  B000   TAB1  0B1D   TAB2  0B32   TAB4  0B9C   TAB5  0BC3
TAB6  0BC9   TABB  0BD1   PROT  0BE7   TTY0  0BED   TTY1  0BF2
TTYI  0BF5   TTCR  0BF8   MON   0BFB   RANT  0BFE   TxTB  0C00
TxTE  1300   VARN  1300   STRH  1336   BUFF  1337   SKED  1381
STAK  1400
```