BASIC-FORTH

A poor man's FORTH computer, and an invitation to FORTH language

by C.H. Ting

C. H. Ting, c/o NDT Technology Laboratory, P. O. Box 504, Sunnyvale, CA

My favorite definition of the FORTH language is "a programming language by which a programmer creates a set of instructions as the solution of his programming problem." The fact that FORTH allows and encourages the programmer to extend and modify its instruction set makes this language unique and exciting, and consequently attracts a host of religiously devoted followers. However, there are still two factors impeding FORTH's even wider growth and dissemination; the peculiar syntax using postfix (reverse Polish) notation, and the expense of a computer which can host a FORTH operating system (above $2000). Such a computer must have at least one diskette drive for bootstrapping and program storage.

There are more than 300,000 small BASIC computers in this country with about 8K bytes of RAM memory and cassette recorder as the only means for program and data storage. Obviously, a FORTH system written entirely in the BASIC language would have definite appeal to owners of these BASIC computers. BASIC-FORTH is designed for these computers with minimum resources. It fully utilizes the facilities already resident in the BASIC operating system to convert the BASIC computer into a dictionary-based FORTH stack computer. BASIC-FORTH's vocabulary includes 12 regular FORTH stack instructions, and 17 BASIC functions converted into FORTH instructions. In addition, the control structure instructions such as IF, ELSE, THEN, BEGIN, UNTIL, DO, and LOOP are also included. These instructions and their stack effects are listed in Table 1.

The bulkiest part of this program is the NUCLEUS which contains the DICTIONARY, a linked list of primitive FORTH instructions coded in BASIC (lines 900 to 1522). At the end of the dictionary is the number conversion routine (lines 1600 to 1608). The operating system of this FORTH system is the TEXT INTERPRETER (lines 30 to 138), which parses lines of input as instructions, and executes the instructions in sequence by first searching the dictionary. An instruction found in the dictionary is immediately executed. If the instruction cannot be located in the dictionary, an attempt is made to convert it into a number and the resulting number is pushed onto the data stack S. If the instruction is neither a member of the dictionary nor a number, an error condition is generated, forcing the execution of line 29. This action aborts the current line of

Table 1
BASIC-FORTH Instructions

Stack Instructions

R?		Type out all stack data.
.	(n ---)	Type out the top number on stack and remove it.
DUP	(n--n n)	Duplicate top of stack.
DROP	(n--)	Discard top of stack.
SWAP	(n1 n2--n2 n1)	Exchange top two stack items.
OVER	(n1 n2--n1 n2 n1)	Make copy of second item on top.
ROT	(n1 n2 n3--n2 n3 n1)	Rotate third item to top.
PICK	(n1--n2)	Copy n1th item to top. 1 PICK=DUP, 2 PICK=OVER.
ROLL	(n--)	Rotate nth item to top. 2 ROLL=SWAP
R@	(--n)	Copy top of return stack to data stack.
>R	(n--)	Move top item to return stack.
R>	(--n)	Retrieve item from return stack.

Arithmetic, Logical, and Functional Instructions

*	(n1 n2--prod)	Multiply.
/	(n1 n2--quot)	Divide.
+	(n1 n2--sum)	Add.
-	(n1 n2--diff)	Subtract.
ABS	(n --|n|)	Absolute value of n.
ATN	(n--arctan n)	Arctangent of n.
COS	(n--cos n)	Cosine of n.
EXP	(n--exp n)	e to the nth power.
INT	(n--integer n)	Integer part of n.
LOG	(n--ln n)	Natural logarithm of n.
RND	(--n)	Leave a random number between 0 and 1.
SGN	(n--sign)	Leave 1, 0, or -1 depending on sign of n.
SIN	(n--sin n)	Sine of n.
SQR	(n--√n)	Square root of n.
TAN	(n--tan n)	Tangent of n.
↑	(n1 n2--n3)	Exponentiation.
=	(n1 n2--n3)	True if n1 is equal to n2.
>	(n1 n2--n3)	True if n1 is greater than n2.
<	(n1 n2--n3)	True if n1 is less than n2.

Control Instructions

IF	(n--)	If n is true, continue execution. Otherwise, skip to ELSE or THEN.
ELSE		Skip to THEN.
THEN		Continue execution.
BEGIN		Save instruction pointer on return stack.
UNTIL	(n--)	Branch to BEGIN if n is false; otherwise continue.
DO	(n1 n2--)	Save instruction pointer, loop limit n1, and loop counter n2 on return stack.
LOOP		Increment loop counter, if equal or greater than loop limit, branch back to DO. Otherwise, discard three items on return stack and continue.

instructions, prints out the offending instruction, and restarts the text interpreter from line 30.

It is difficult to implement the colon definitions or other high-level FORTH instructions, because the text interpreter cannot easily extend the dictionary. Between lines 300 and 900, some simulated high-level FORTH instructions are coded as examples. The equivalent FORTH instructions are:

```
: SQUARE    DUP * ;
: CUBE      DUP SQUARE * ;
```

The content of a colon instruction, that is, the names of other predefined instructions in the dictionary, is implemented as a string B$, which is concatenated to the calling string I$ currently under interpretation. The nesting level pointer K is incremented to save the current interpretative pointer L(K) in the calling string. L(K) will be the point of return when the called instruction is completely executed. Since L(n) is defined as an array of dimension 10, this program allows ten levels of nesting within high-level FORTH instructions. The depth of nesting can be increased by re-dimensioning L and L0.

The data stack S(n) can hold 40 numbers, and the return stack R(n) can hold 20 numbers. The dimensions of these stacks can also be modified. The return stack is used only by the following instructions: >R, R>, R@, BEGIN, UNTIL, DO, and LOOP. The loop instructions use the return stack to hold pointers for backward branchings. The functions served by the return stack in these loop instructions are different from that of regular FORTH. The user must exercise caution in using the return stack.

All instructions are interpreted and immediately executed without exception. The IF-ELSE-THEN construct is of some interest in its implementation. IF examines the top of the data stack. If the number is true, execution continues to ELSE. ELSE skips all instructions up to THEN, and continues execution after THEN. If the number is false, IF continues execution after the next ELSE or THEN – whichever is encountered first. With this scheme, unlike regular FORTH, IF-ELSE-THEN structures cannot be nested into each other. However, IF-ELSE-THEN structures can be nested into other loop structures.

The program listings are shown on page 40. This program was coded in the Extended BASIC on a NOVA-3 computer made by Data General Co. It would be very easy to modify this program for execution on small BASIC computers. The only problem which might arise is in the area of string functions used to implement the text interpreter. Some modification may be needed depending on how string variables are handled and how substrings are extracted and concatenated. The program can be extended or modified to suit specific applications. The area between line 300 and 900 is reserved for additional high-level or low-level instructions. The FORTH interpreter runs slowly under BASIC, but not terribly so. Its advantage over regular FORTH systems is that all numbers are floating-point numbers, and most of the transcendental functions in BASIC are available for interactive execution.

The main purpose of this BASIC-FORTH is educational – to demonstrate the basic principles involved in a virtual FORTH computer such as the dictionary, the stacks, and the interpreter. FORTH as a programming language can be implemented on different computers, even on top of BASIC as in this case. The program itself is also useful to BASIC programmers designing small control systems using these low-cost BASIC computers.

(LISTING ON PAGE 40)

BASIC FORTH (Listing)

```
0001 REM **********   BASIC-FORTH  V. 3 ****************
0002 DIM S(40),R(20),L(10),L0(10),I$(800)
0003 DIM B$(80)
0004 PRINT 'BASIC-FORTH  V.3'
0020 REM  N IS SP, M IS RP, K IS IP, AND L IS W.
0024 ON ERR THEN GOTO 0029
0026 ON ESC THEN STOP
0028 GOTO 0030
0029 PRINT A$,' ?'
0030 LET M=0
0032 LET N=0
0050 REM ********  TEXT INTERPRETER   **********
0060 LET K=1
0062 INPUT I$
0064 LET L1=0
0070 LET L(K)=L1
0072 LET L0(K)=LEN(I$)
0074 LET L1=L0(K)
0100 IF N<0 THEN GOTO 0106
0104 GOTO 0110
0106 PRINT 'STACK EMPTY'
0108 GOTO 0030
0110 LET L(K)=L(K)+1
0112 IF L(K)>L0(K) THEN GOTO 0132
0114 LET B$=I$(L(K),L(K))
0116 IF B$=' ' THEN GOTO 0110
0118 LET A$=B$
0120 LET L(K)=L(K)+1
0121 IF L(K)>L0(K) THEN GOTO 0130
0122 LET B$=I$(L(K),L(K))
0124 IF B$=' ' THEN GOTO 0130
0126 LET A$=A$,B$
0128 GOTO 0120
0130 GOTO 0200
0132 IF K<2 THEN GOTO 0060
0134 LET K=K-1
0135 LET I$=I$(1,L0(K))
0136 LET L1=L0(K)
0138 GOTO 0110
0200 REM **********  DICTIONARY  **********
0210 REM   300-900 :: HIGH LEVEL DEFINITIONS
0300 IF A$<>'SQUARE' THEN GOTO 0310
0302 LET B$='DUP *'
0304 LET I$=I$,B$
0306 LET K=K+1
0308 GOTO 0070
0310 IF A$<>'CUBE' THEN GOTO 0320
0312 LET B$='DUP SQUARE *'
0314 LET I$=I$,B$
0316 LET K=K+1
0318 GOTO 0070
0320 IF A$<>'TEST' THEN GOTO 0330
0322 LET B$='DO PI 10 / R@ * SIN . LOOP'
0324 LET I$=I$,B$
0326 LET K=K+1
0328 GOTO 0070
0330 REM
0900 REM  **********  LOW LEVEL DEFINITIONS-- NUCLEUS **********
0902 IF A$<>'+' THEN GOTO 0910
0904 LET N=N-1
```

continued

```
0906 LET S(N)=S(N)+S(N+1)
0908 GOTO 0100
0910 IF A$<>"-" THEN GOTO 0920
0912 LET N=N-1
0914 LET S(N)=S(N)-S(N+1)
0916 GOTO 0100
0920 IF A$<>"*" THEN GOTO 0930
0922 LET N=N-1
0924 LET S(N)=S(N)*S(N+1)
0926 GOTO 0100
0930 IF A$<>"/" THEN GOTO 0940
0932 LET N=N-1
0934 LET S(N)=S(N)/S(N+1)
0936 GOTO 0100
0940 IF A$<>"ABS" THEN GOTO 0950
0942 LET S(N)=ABS(S(N))
0944 GOTO 0100
0950 IF A$<>"ATN" THEN GOTO 0960
0952 LET S(N)=ATN(S(N))
0954 GOTO 0100
0960 IF A$<>"COS" THEN GOTO 0970
0962 LET S(N)=COS(S(N))
0964 GOTO 0100
0970 IF A$<>"EXP" THEN GOTO 0980
0972 LET S(N)=EXP(S(N))
0974 GOTO 0100
0980 IF A$<>"INT" THEN GOTO 0990
0982 LET S(N)=INT(S(N))
0984 GOTO 0100
0990 IF A$<>"LOG" THEN GOTO 1000
0992 LET S(N)=LOG(S(N))
0994 GOTO 0100
1000 IF A$<>"RND" THEN GOTO 1010
1002 LET S(N)=RND(-N)
1004 GOTO 0100
1010 IF A$<>"SGN" THEN GOTO 1020
1012 LET S(N)=SGN(S(N))
1014 GOTO 0100
1020 IF A$<>"SIN" THEN GOTO 1030
1022 LET S(N)=SIN(S(N))
1024 GOTO 0100
1030 IF A$<>"SQR" THEN GOTO 1040
1032 LET S(N)=SQR(S(N))
1034 GOTO 0100
1040 IF A$<>"TAN" THEN GOTO 1050
1042 LET S(N)=TAN(S(N))
1044 GOTO 0100
1050 IF A$<>"^" THEN GOTO 1060
1052 LET N=N-1
1054 LET S(N)=S(N)^S(N+1)
1056 GOTO 0100
1060 IF A$<>"S?" THEN GOTO 1070
1062 FOR I=1 TO N
1064   PRINT S(N-I+1)
1066 NEXT I
1068 GOTO 0100
1070 IF A$<>"." THEN GOTO 1080
1071 IF N<1 THEN GOTO 0106
1072 PRINT S(N)
1074 LET N=N-1
1076 GOTO 0100
```

(Continued on next page)

```
1080 IF A$<>"DUP" THEN GOTO 1090
1082 LET N=N+1
1084 LET S(N)=S(N-1)
1086 GOTO 0100
1090 IF A$<>"DROP" THEN GOTO 1100
1092 LET N=N-1
1094 GOTO 0100
1100 IF A$<>"SWAP" THEN GOTO 1110
1102 LET S(N+1)=S(N-1)
1104 LET S(N-1)=S(N)
1106 LET S(N)=S(N+1)
1108 GOTO 0100
1110 IF A$<>"OVER" THEN GOTO 1120
1112 LET N=N+1
1114 LET S(N)=S(N-2)
1116 GOTO 0100
1120 IF A$<>">R" THEN GOTO 1130
1122 LET M=M+1
1124 LET R(M)=S(N)
1126 LET N=N-1
1128 GOTO 0100
1130 IF A$<>"R>" THEN GOTO 1140
1132 LET N=N+1
1134 LET S(N)=R(M)
1136 LET M=M-1
1138 GOTO 0100
1140 IF A$<>"R@" THEN GOTO 1200
1142 LET N=N+1
1144 LET S(N)=R(M)
1146 GOTO 0100
1200 REM **********  CONTROL STRUCTURES  *********
1202 IF A$<>"=" THEN GOTO 1210
1203 LET N=N-1
1204 IF S(N)=S(N+1) THEN GOTO 1207
1205 LET S(N)=0
1206 GOTO 0100
1207 LET S(N)=1
1209 GOTO 0100
1210 IF A$<>">" THEN GOTO 1220
1212 LET N=N-1
1214 IF S(N)>S(N+1) THEN GOTO 1217
1215 LET S(N)=0
1216 GOTO 0100
1217 LET S(N)=1
1218 GOTO 0100
1220 IF A$<>"<" THEN GOTO 1230
1222 LET N=N-1
1223 IF S(N)<S(N+1) THEN GOTO 1227
1224 LET S(N)=0
1225 GOTO 0100
1227 LET S(N)=1
1228 GOTO 0100
1230 IF A$<>"IF" THEN GOTO 1250
1231 LET N=N-1
1232 IF S(N+1) THEN GOTO 0100
1233 FOR I=L(K) TO LO(K)-3
```

(Continued)

```
1234   LET B$=I$( I,I+3)
1235   IF B$='ELSE' THEN GOTO 1240
1236   IF B$='THEN' THEN GOTO 1240
1237 NEXT I

1238 PRINT 'IF?'
1239 GOTO 0030
1240 LET L(K)=I+4
1241 GOTO 0100
1242 GOTO 0100
1250 IF A$<>'ELSE' THEN GOTO 1260
1252 GOTO 1233
1260 IF A$<>'THEN' THEN GOTO 1270
1262 GOTO 0100
1270 IF A$<>'BEGIN' THEN GOTO 1280
1272 LET M=M+1
1274 LET R(M)=L(K)
1276 GOTO 0100
1280 IF A$<>'UNTIL' THEN GOTO 1300
1282 LET N=N-1
1283 IF S(N+1) THEN GOTO 1288
1284 IF S(N+1) THEN GOTO 0100
1286 LET L(K)=R(M)
1287 GOTO 0100
1288 LET M=M-1
1289 GOTO 0100
1300 IF A$<>'DO' THEN GOTO 1320
1302 LET M=M+1
1304 LET R(M)=L(K)
1305 LET M=M+1
1306 LET R(M)=S(N-1)
1308 LET M=M+1
1309 LET R(M)=S(N)
1310 LET N=N-2
1312 GOTO 0100
1320 IF A$<>'LOOP' THEN GOTO 1340
1322 LET R(M)=R(M)+1
1324 IF R(M-1)>R(M) THEN GOTO 1330
1326 LET M=M-3
1328 GOTO 0100
1330 LET L(K)=R(M-2)
1332 GOTO 0100
1340 REM *******  CONSTANTS  *********
1500 IF A$<>'PI' THEN GOTO 1510
1502 LET N=N+1
1504 LET S(N)=3.14159
1506 GOTO 0100
1510 IF A$<>'0' THEN GOTO 1520
1512 LET N=N+1
1514 LET S(N)=0
1516 GOTO 0100
1520 IF A$<>'STOP' THEN GOTO 1600
1522 STOP
1600 REM **********  NUMBER  **********
1604 LET N=N+1
1606 LET S(N)=VAL(A$)
1608 GOTO 0100
```

End Listing