FLOPTRAN-IV: A Tiny Compiler

Mark Zimmermann
9410 Woodland Dr
Silver Spring MD 20910

These "programmers' programs" allowed a user to write in a language that is more compact and meaningful for humans than machine language.

In the beginning, computers were programmed in machine language. To execute a simple equation, such as A=B, a user had to write dozens of instructions. He had to tell the machine, "load the accumulator with the first byte of the number B," and "store the accumulator in the memory location assigned to the first byte of A," *ad nauseam*. A more complicated equation like C=LOG(D) required hundreds or even thousands of instructions! It's no wonder, therefore, that among the first programs developed were programs to make it easier to write other programs. These "programmers' programs" allowed a user to give the computer instructions in a higher-level language, one that is more compact and meaningful for humans than machine language.

High-level computer languages are nonsense to a machine until they have been translated into fundamental commands that the system can understand and execute. There are two principal methods used to perform this translation: interpretation and compilation. An *interpreter* is a machine-language program that scans through a higher-level program, line by line. As each statement is encountered, the interpreter figures out which operations are necessary to execute that statement, then goes ahead and performs the operations. The interpreter then moves on to the next statement, translates it, and executes it. This process is repeated, with each statement being interpreted (or reinterpreted) *every* time it is executed.

A compiler acts rather differently. Like an interpreter, it scans through a high-level program and figures out what fundamental operations are necessary to execute each line. But, instead of performing those operations, a compiler just writes out (ie: onto a cassette tape drive) the sequence of machine-language instructions that do the job when executed. An interpreter executes a program statement by statement, as it interprets it, so that the interpreter and the program execute concurrently. But, when a compiler is finished with a program, the result is a machine-language program that represents the higher-level program; the machine-language program must be loaded back into the computer and executed before the problem being programmed is solved.

The two approaches, compilation and interpretation, have different strengths and weaknesses. An interpreter, interpreting and executing line by line, can stop at the first sign of an error and give a helpful and specific error message. The interpretation can also be interrupted at any point (to be later resumed) allowing the human operator to look at the partial results of the calculation and see whether everything is going as planned. This interactive technique makes program development and debugging relatively easy.

On the other hand, when an interpreter has finished running a job, the machine is no "smarter" than when it began. Executing the program a second time will take just as long as it did the first time. In fact, if there is looping within the program so that some lines are executed more than once, these lines are completely reinterpreted, as if they had never been seen before, each time they are encountered. This redundant interpretation wastes a lot of computer time, making interpreted programs somewhat slower than compiled ones.

However, once a compiler has finished with a program, the result is a pure machine-language series of instructions that can be executed without further translation. Although this results in more vague error messages, the advantage to compilation is speed. Depending on the problem, a compiled program typically executes several times faster than an interpreted one—sometimes, the speed advantage is as large as a factor of 100. Once a program has been compiled, it never needs to be compiled again.

The output of a compiler does tend to be larger than the original source program (which the interpreter uses), so that interpreters may seem to have the advantage in terms of memory usage. But, when one remembers that the interpreter must be continuously resident in the computer's memory while the program is running, whereas the compiler can be removed after the program has been compiled, the apparent memory advantage of interpretation over compilation shrinks and often vanishes.

Operator	Meaning
+	add
−	subtract
*	multiply
/	divide
↑	exponentiation
ABS	absolute value
ATN	arctangent
COS	cosine
EXP	powers of e
FRE	memory available
INT	integer portion of argument
LOG	base-e logarithm
PEEK	look at contents of memory
POS	position of cursor
RND	random number
SGN	sign of argument
SIN	sine
SQR	square root
TAN	tangent

Table 1: *Available operators and functions in FLOPTRAN-IV.*

Some languages, such as FORTRAN, are usually compiled and are hardly ever interpreted. Other languages, like BASIC, are generally interpreted. Many small-computer systems come with a built-in or easily-loaded BASIC interpreter, but to get a compiled language, it is necessary to pay hundreds of dollars (or more) for extra memory, disk storage, and a compiler.

But there is no reason why a language like BASIC should not be compiled instead of interpreted. Also, there is no reason to ignore or throw away the wonderful resource that a typical BASIC in read-only memory represents. In those read-only memories are hidden dozens of useful, thoroughly tested, efficient machine-language subroutines; it would be a shame not to take full advantage of them.

What I've done is to write a tiny compiler for a language I call FLOPTRAN-IV, which stands for *FLOating-Point TRANslator, version 4*. The compiler is written in Microsoft (PET) BASIC and is presented in full in listing 1. As written and debugged, the FLOPTRAN-IV compiler runs on an original-version 8 K-byte Commodore PET. The new PETs, which have a different set of read-only memory chips, have moved many of the subroutine entry points so that, to run FLOPTRAN on such a machine, it will be necessary to find the new addresses and change them as they appear in the program of listing 1.

Other 6502-microprocessor machines using Microsoft BASIC, such as the Apple and Ohio Scientific systems, will require more extensive changes in the FLOPTRAN compiler in order to make it operational, although the overall structure of the compiler should still be applicable. In fact, the idea of a tiny compiler that uses the resident BASIC subroutines to do the bulk of its work is extremely general and applicable to systems using *any* microprocessor. Once I had thought of the concept, it took me only a couple of days to get the compiler up and running; anyone with an intimate knowledge of his personal system should be able to do as well.

Introduction to FLOPTRAN-IV

FLOPTRAN, the floating-point translator, is so named because it is heavily oriented toward floating-point mathematical operations. For compatibility with the PET's read-only memory routines, it uses the standard Microsoft format for storing numbers: five 8-bit bytes, with 1 byte for the exponent (power of 2) and 4 bytes for the fractional part (or mantissa) of the number and its sign. Built into FLOPTRAN are the functions given in table 1; the ability to call a user-defined machine-language function USR is also included, although it may be used infrequently. [*Several articles on computer representation of floating-point numbers may be found in the book* Numbers in Theory and Practice, *which is edited by Blaise W Liffick and is available from BYTE Books....*RSS]

A FLOPTRAN program looks like a BASIC program and, with one particular exception, it *is* a legal BASIC program. That exception is the use of the RETURN statement to end the program, as well as to signal a return-from-subroutine. This is done so that the compiled FLOPTRAN program ends with the 6502 instruction RTS (return-from-subroutine); thus, after the compiled program is run using the SYS command, control will return to the PET operating system.

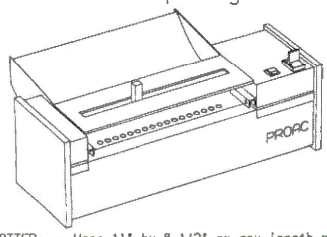
Because FLOPTRAN is essentially BASIC, it is easy to test and debug a FLOPTRAN program by using the BASIC interpreter built into the PET. When the program has been fully debugged, the compiler program need only be executed once; thereafter, the compiled machine-language program can be executed directly.

Note that the PET's Microsoft BASIC interpreter stores the reserved words (ie: keywords) of the BASIC language as single bytes in memory: GOTO is stored as the single-

Text continued on page 212

LEAPAC SERVICES

MAURO PROAC MP-250 PLOTTER - $695
with L2D Plot package - $795
with L2D & L3P packages - $950



LEAPAC SERVICES

MAURO PLOTTER - Uses 11" by 8-1/2" or any length paper. Resolution is 200 steps per inch. 0.005" tracking error. Mauro X-Y vector software with pen control is available for 8080/Z80, 6502, & 6800 micro-processors. Requires 3 bits of a parallel output port. APPLE, TRS-80, and RS232 interfaces are available as I/O Options.

LEAPAC SOFTWARE - Supports complete 2D & perspective plotting, including ASCII and curve generation. Available as relative linking libraries (L80) for MICROSOFT compatible software products, FORTRAN-80, COBOL-80, COMPILER BASIC, and MAGRO-80 in CP/M compatible files on 8" IBM-3740 disks or 5-1/4" NORTHSTAR formatted disks.

L2D, X-Y plot package. Contains over 30 entries, including CALCOMP compatible calls such as PLOT and WHERE.

L3P Perspective plot package. Contains over 70 entries. Capable of ZOOMING, FLY-BYS, ANIMATION, and much more. (See self portrait above).

LEAPAC SERVICES (916) 381-1717
8245 MEDITERRANEAN WAY SACRAMENTO CA 95826
DEALER INQUIRIES ARE INVITED

CP/M is a registered trade mark of Digital Research, Inc.
MICROSOFT is a trade mark of MICROSOFT, Inc.CP
CALCOMP is a trade mark of California Computer Products, Inc.

Circle 132 on inquiry card.

How to Use the FLOPTRAN-IV Compiler

To write and compile a program:

1) *Load the FLOPTRAN-IV compiler program of listing 1.*
2) *Write the program to be compiled in FLOPTRAN (see summary of syntax in table 2) giving statements line numbers in the range 0 thru 255, and terminating the FLOPTRAN program with a RETURN statement instead of END.*
3) *Optional: you may want to test the uncompiled FLOPTRAN program using the BASIC interpreter; if you change the terminal RETURN to an END or STOP for that purpose, be sure to change it back before compiling.*
4) *To compile, place a blank cassette in the PET tape drive and type RUN 50000. The compiled program will be written to the cassette.*

To load and execute a compiled program:

1) *Type NEW, then load and run the program in listing 3. This loads the compiled program's origin (the first thing FLOPTRAN writes to tape), then loads the program until it hits the "—1" signal that marks the end of the first pass. It then loads the addresses of JMP or JSR targets to be corrected until it hits the final "—1" marking the end of the second pass; this is the end of the data file.*
2) *To run the compiled program now in memory, type SYS(nnnn), where nnnn is the decimal number given in the "BEGIN AT nnnn" message from listing 3.*

Note: FLOPTRAN-IV as presently designed is for old PET read-only-memory computers only. Many addresses will have to be changed for this compiler to work with new PETs.

Listing 1: *FLOPTRAN-IV compiler program for the original-version PET. This program will convert a FLOPTRAN-IV program (actually a subset of BASIC) to an executable machine-code program, which is written to the PET cassette tape drive to be loaded at a later time. When typing this listing into a PET, all the statements associated with one line number are to be typed on the same line, separated by colons.*

```
49000 X=                                          ;;;leave at least 60 blanks here
     :RETURN                                      ;;;to be filled in during a 'LET' operation

49100 TP=TP+1                                     ;;;subroutine to get next nonblank character
     :CH=PEEK(TP)                                 ;;;and return it in CH
     :IF CH=32 GOTO 49100

49120 RETURN

49190 IF (CH>64) AND (CH<91) GOTO 49300           ;;;64<CH<91 means alphabetic, legal variable name

49200 PRINT                                       ;;;abort if unexpected character found
     :PRINT "***ERROR IN LINE";LI;"***"
     :GOTO 61000

49300 Z=5*(CH-65)+OV                              ;;;return the address (5 byte location) of the variable
     :ZH=INT(Z/256)                               ;;;'CH'...high and low parts of address in ZH & ZL
     :ZL=Z-256*ZH
     :RETURN

49400 PRINT#1,169                                 ;;;LDA #ZL
     :PRINT#1,ZL
     :PRINT#1,160                                 ;;;LDY #ZH
     :PRINT#1,ZH

49420 PRINT#1,32                                  ;;;JSR $DA74 with A & Y  set up as above
     :PRINT#1,116                                 ;;;transfers the value of the variable being
     :PRINT#1,218                                 ;;;pointed at to the primary accumulator for floating-
     :PC=PC+7                                     ;;;-point operations, 'PRI'
     :RETURN                                      ;;;add 7 bytes to the program-counter & return

49600 DATA 11,219,158,219,42,219,0,0,98,210,133,210   ;;;data for floating-point ROM subroutine calls, 2-byte pairs
                                                  ;;;SGN,INT,ABS,USR,FRE,POS
49640 DATA 36,222,69,223,191,216,160,222,158,223,165,223
                                                  ;;;SQR,RND,LOG,EXP,COS,SIN
49680 DATA 238,223,72,224,230,214,60,215,40,215,253,216,228,217,46,222
                                                  ;;;TAN,ATN,PEEK,+,-,*,/,^
```

Listing 1 continued on page 202

Circle 133 on inquiry card. →

Listing 1 continued:

```
49720 FOR X=0 TO 19
     :READ ML%(X,0),ML%(X,1)                    ;;;load addresses into ML% array, low byte, then high
     :NEXT X

49740 FOR X=0 TO 255
     :LN%(X)=-32767                             ;;;initialize LN% array of line numbers so that nonexistent
     :NEXT X                                    ;;;lines will be flagged later
     :RETURN

50000 DIM X,LN%(255),GO%(127,1),ML%(19,1)       ;;;RUN 50000 to enter program here & compile code
     :P=1025                                    ;;;dimension X first so make it easy to find later
     :GC=0                                      ;;;initialize pointer P & GOTO/GOSUB counter GC

50040 L=PEEK(P+2)+256*PEEK(P+3)                 ;;;look up line number & scan
     :IF L<>49000 THEN P=PEEK(P)+256*PEEK(P+1)  ;;;to locate subroutine 49000 to be used in 'LET'
     :GOTO 50040                                ;;;assignments during later compilation

50100 XL=P+6                                    ;;;store location  of first space after '=' in line 49000 in XL
     :GOSUB 49600                               ;;;initialize ML% and LN% arrays
     :INPUT "COMPILED DATA FILE NAME";C$
     :OPEN 1,1,1,C$                             ;;;get file name & open data file for compiled program

50120 INPUT "ORIGIN FOR CODE";OC                ;;;OC stores starting location for compilation
     :PRINT#1,OC                                ;;;write it to tape to begin data file
     :PC=OC                                     ;;;initialize program counter PC

50140 INPUT "VARIABLE TABLE ORIGIN";OV          ;;;variables occupy 26*5 contiguous locations beginning at OV
     :TP=1028                                   ;;;initialize text pointer to first line number of BASIC program

50200 LI=PEEK(TP-1)+256*PEEK(TP)                ;;;put line number in LI
     :IF LI>255 GOTO 60000                      ;;;compiling program has lines in range 0-255----exit when finished

50220 LN%(LI)=PC-32767                          ;;;store current PC in LN%(LI) to be used in GOTOs or GOSUBs later
     :PRINT "COMPILING LINE";LI                 ;;;announce to screen what line is being compiled now

50240 GOSUB 49100                               ;;;get next nonblank character in CH

50260 IF CH=0 THEN TP=TP+4                      ;;;'0' is end-of-line marker----move text pointer forward and
     :GOTO 50200                                ;;;loop to get next line number

50280 IF CH=58 GOTO 50240                       ;;;'58' is ':'----loop back to allow multiple statements/line

50300 IF CH=136 GOTO 58000                      ;;;'136' is 'LET'

50320 IF CH=153 GOTO 59000                      ;;;'153' is 'PRINT'

50340 IF CH=133 GOTO 59500                      ;;;'133' is 'INPUT'

50360 IF CH=139 GOTO 58500                      ;;;'139' is 'IF'

50380 IF (CH=137) OR (CH=141) GOTO 58700        ;;;'137' is 'GOTO' and '141' is 'GOSUB'

50400 IF CH=143 GOTO 56500                      ;;;'143' is 'REM'

50420 IF CH=142 THEN PRINT#1,96                 ;;;'142' is 'RETURN', the 6502 'RTS' instruction, 96 decimal
     :PC=PC+1                                   ;;;'RETURN' also ends the compiled program & returns to BASIC after
     :GOTO 50240                                ;;;a 'SYS' command starts machine-language

50440 GOSUB 49190                               ;;;if none of the above, it better be a variable name----check it


     :VL=ZL                                     ;;;and store the storage address in VL, VH
     :VH=ZH

50460 GOSUB 49100                               ;;;get next character
     :IF CH<>178 GOTO 49200                     ;;;it had better be an '=' sign...abort if not!

50480 GOSUB 49100                               ;;;next character
     :IF (CH>179) AND (CH<195) GOTO 51000       ;;;179<CH<195 means a monadic function

50500 IF CH=171 GOTO 55600                      ;;;'171' is a '-' sign

50520 GOSUB 49190                               ;;;otherwise, it better be a variable name....
     :WL=ZL                                     ;;;store the address of the variable in WL, WH
     :WH=ZH
```

Listing 1 continued on page 204

Listing 1 continued:

```
50540 GOSUB 49100
     :IF (CH=0) OR (CH=58) GOTO 58300          ;;;if 'end-of-line' or ':', equation is of the form 'V=W'

50560 IF (CH<170) OR (CH>174) GOTO 49200       ;;;otherwise, expect +,-,*,/,^ and abort if not

50580 FU=CH-155                                ;;;FU now records what operator it was
     :GOSUB 49100                              ;;;get next character
     :GOSUB 49190                              ;;;better be variable name...address in ZL, ZH
     :GOSUB 49400                              ;;;write out code to transfer it to primary accumulator

50660 PRINT#1,169                              ;;;LDA #WL
     :PRINT#1,WL
     :PRINT#1,160                              ;;;LDY #WH
     :PRINT#1,WH
     :PC=PC+4                                  ;;;count the bytes and increment program counter

50700 IF (FU=15) OR (FU=17) GOTO 51080         ;;;take shortcut for '+' or '*' functions

50720 PRINT#1,32                               ;;;JSR $D95E transfers 'W' to secondary accumulator
     :PRINT#1,94                               ;;;and sets up for dyadic operation subroutine call
     :PRINT#1,217
     :PC=PC+3                                  ;;;count the bytes...
     :GOTO 51080                               ;;;jump down to output code to do the operation & store result

51000 FU=CH-180                                ;;;arrive here to perform a monadic function
     :GOSUB 49100
     :IF CH<>40 GOTO 49200                     ;;;next character better be a '(', else abort!

51040 GOSUB 49100
     :GOSUB 49190                              ;;;get argument  of function, address in ZL, ZH
     :GOSUB 49400                              ;;;fetch it to primary floating accumulator

51060 GOSUB 49100
     :IF CH<>41 GOTO 49200                     ;;;expect to see a ')' here!!!

51080 PRINT#1,32                               ;;;JSR to ROM function
     :PRINT#1,MLX(FU,0)
     :PRINT#1,MLX(FU,1)
     :PC=PC+3

51100 PRINT#1,162                              ;;;LDX #VL
     :PRINT#1,VL
     :PRINT#1,160                              ;;;LDY #VH
     :PRINT#1,VH

51120 PRINT#1,32                               ;;;JSR $DAA6 with target in X & Y registers
     :PRINT#1,166                              ;;;to store result of calculation from primary accumulator
     :PRINT#1,218
     :PC=PC+7
     :GOTO 50240                               ;;;back to top of main loop

55600 GOSUB 49100                              ;;;handle monadic '-' operator
     :GOSUB 49190
     :GOSUB 49400                              ;;;transfer number to be negated to primary floating accumulator
     :PRINT#1,169                              ;;;LDA #$FF
     :PRINT#1,255

55660 PRINT#1,69                               ;;;EOR $B5 (sign storage for primary accumulator)
     :PRINT#1,181
     :PRINT#1,133                              ;;;STA $B5....now sign is flipped
     :PRINT#1,181
     :PC=PC+6
     :GOTO 51100                               ;;;go store negated number in target

56500 GOSUB 49100                              ;;;handle 'REM' statement
     :IF CH>0 GOTO 56500                       ;;;loop until end-of-line encountered

56540 GOTO 50260                               ;;;back to top of main loop

58000 GOSUB 49100                              ;;;handle 'LET' assignment operation
     :GOSUB 49190                              ;;;get address for result to be stored into
     :GOSUB 49100                              ;;;this character better be an '=' sign
     :VL=TP+1                                  ;;;save pointer to first character after '=' in VL
     :IF CH<>178 GOTO 49200                    ;;;abort if it wasn't '='

58080 GOSUB 49100
     :IF (CH<>58) AND (CH<>0) GOTO 58080       ;;;scan until ':' or 'end-of-line' terminates assignment
```

Listing 1 continued on page 206

Listing 1 continued:

```
58100 IF TP-VL>60 THEN PRINT "LINE TOO LONG!"        ;;;subroutine 49000 only has room for 60 characters----sorry!
     :GOTO 49200

58120 FOR L=VL TO TP-1                               ;;;take the whole expression to be evaluated for the 'LET'
     :WL=PEEK(L)
     :POKE XL+L-VL,WL                                ;;;and POKE it into subroutine 49000
     :NEXT L

58160 GOSUB 49000                                    ;;;go evaluate the expression!----result returns in variable X



     :FOR L=VL TO TP-1
     :POKE XL+L-VL,32                                ;;;and clean up (fill with blanks again) subroutine 49000
     :NEXT L

58180 PRINT#1,24                                     ;;;CLC
     :PRINT#1,144                                    ;;;BCC .+5   (a forced branch over the following 5 bytes)
     :PRINT#1,5
     :PC=PC+3
     :VL=256*PEEK(125)+PEEK(124)+2                   ;;;point VL to the place where X is stored now

58220 FOR L=VL TO VL+4                               ;;;fetch all 5 bytes of the current value of X
     :PRINT#1,PEEK(L)                                ;;;and write them into the compiled file
     :NEXT L
     :WH=INT(PC/256)                                 ;;;store their location in the compiled code in WH, WL
     :WL=PC-256*WH

58240 VL=ZL                                          ;;;put the address for the result of the 'LET' to be stored into
     :VH=ZH                                          ;;;in VL, VH, and fall into the 'assignment' handling routine
     :PC=PC+5

58300 PRINT#1,162                                    ;;;handle simple assignments of the form 'V=W' here
     :PRINT#1,4                                      ;;;LDX #4
     :PRINT#1,189                                    ;;;LDA W,X
     :PRINT#1,WL
     :PRINT#1,WH

58340 PRINT#1,157                                    ;;;STA V,X
     :PRINT#1,VL
     :PRINT#1,VH
     :PRINT#1,202                                    ;;;DEX

58360 PRINT#1,16                                     ;;;BPL .-9
     :PRINT#1,247
     :PC=PC+11                                       ;;;transfer finished...count bytes & back to main loop top
     :GOTO 50260

58500 GOSUB 49100                                    ;;;handle 'IF' statements
     :GOSUB 49190                                    ;;;get variable following the 'IF' ---- address in Z
     :PRINT#1,173                                    ;;;LDA Z
     :PRINT#1,ZL
     :PRINT#1,ZH
     :PRINT#1,240                                    ;;;BEQ .+3    (if variable = 0, skip over following 3 bytes)

58580 PRINT#1,3
     :PC=PC+5
     :GOSUB 49100                                    ;;;get next character
     :IF (CH<>137) AND (CH<>141) GOTO49200           ;;;abort if not 'GOTO' or 'GOSUB'----else, fall into following....

58700 IF CH=141 THEN PRINT#1,32                      ;;;handle  'GOSUB' or 'GOTO' statements
     :GOTO 58760                                     ;;;'GOSUB' becomes 'JSR'

58740 PRINT#1,76                                     ;;;and 'GOTO' becomes 'JMP'

58760 GO%(GC,0)=PC-32766                             ;;;store in GO% array the address of the compiled-code's bytes after
     :PRINT#1,0                                      ;;;the opcode, to be filled in later (second-pass)
     :PRINT#1,0                                      ;;;print zeroes for now, to hold the spaces
     :PC=PC+3
     :TL=0                                           ;;;initialize target line number TL

58800 GOSUB 49100
     :IF (CH<48) OR (CH>57) GOTO 58840               ;;;read in ASCII target line number

58820 TL=10*TL+CH-48                                 ;;;and convert it to a number in TL
     :GOTO 58800
```

Listing 1 continued on page 210

Listing 1 continued:

```
58840 GOX(GC,1)=TL                          ;;;record target line number in GOX
      :GC=GC+1                              ;;;increment GOTO/GOSUB counter GC
      :GOTO 50260                           ;;;back to top of main loop

59000 GOSUB 49100                           ;;;handle a 'PRINT' statement
      :IF (CH=0) OR (CH=58) GOTO 59400      ;;;when done with line or statement, print a <CRLF> before quitting

59040 IF (CH=44) OR (CH=59) GOTO 59000      ;;;keep going if ',' or ';' encountered within 'PRINT'

59080 GOSUB 49190                           ;;;get address of variable to be printed
      :GOSUB 49400                          ;;;transfer it to primary accumulator
      :PRINT#1,32                           ;;;JSR $DCAF converts it to ASCII string at top of page 1
      :PRINT#1,175
      :PRINT#1,220

59100 PRINT#1,162                           ;;;LDX #0
      :PRINT#1,0
      :PRINT#1,189                          ;;;LDA $100,X
      :PRINT#1,0
      :PRINT#1,1

59120 PRINT#1,240                           ;;;BEQ .+6   (end of string is marked by a zero)
      :PRINT#1,6
      :GOSUB 59300                          ;;;subroutine outputs code to print accumulator on screen
      :PRINT#1,232                          ;;;INX
      :PRINT#1,208                          ;;;BNE .-$B (always loop back...number is never longer than 16 bytes)
      :PRINT#1,245

59160 PRINT#1,169                           ;;;LDA #32   (ASCII blank space)
      :PRINT#1,32
      :GOSUB 59300                          ;;;print accumulator out
      :PC=PC+21                             ;;;count bytes
      :GOTO 59000                           ;;;and continue with print statement, looping to above

59300 PRINT#1,32                            ;;;JSR $FFD2 prints out accumulator onto screen
      :PRINT#1,210
      :PRINT#1,255

      :RETURN

59400 PRINT#1,169                           ;;;LDA #$D   (<CRLF>)
      :PRINT#1,13
      :GOSUB 59300                          ;;;print it
      :PC=PC+5
      :GOTO 50260                           ;;;back to top of main loop

59500 GOSUB 49100                           ;;;handle 'INPUT' statement
      :IF (CH=0) OR (CH=58) GOTO 50260      ;;;finished when 'end-of-line' or ':'

59540 IF (CH=44) OR (CH=59) GOTO 59500      ;;;keep going if ',' or ';'

59560 GOSUB 49190
      :PRINT#1,169                          ;;;LDA #CH
      :PRINT#1,CH
      :GOSUB 59300                          ;;;print prompting letter on screen before input request

59580 PRINT#1,169                           ;;;LDA #$3F   (ascii '?')
      :PRINT#1,63
      :GOSUB 59300                          ;;;print it

59600 PRINT#1,162                           ;;;LDX #0   (to initialize counter for input characters)
      :PRINT#1,0
      :PRINT#1,32                           ;;;JSR $FFCF  (inputs, with cursor for user to edit)
      :PRINT#1,207
      :PRINT#1,255

59620 PRINT#1,149                           ;;;STA $A,X   (store input stuff in BASIC input buffer)
      :PRINT#1,10
      :PRINT#1,232                          ;;;INX
      :PRINT#1,201                          ;;;CMP #$D   (check for <CRLF>)
      :PRINT#1,13

59660 PRINT#1,208                           ;;;BNE .-$A   (loop and keep inputting)
      :PRINT#1,246
      :GOSUB 59300                          ;;;echo <CRLF>
```

Listing 1 continued on page 212

Listing 1 continued:

```
59680 PRINT#1,169                                   ;;;LDA #0
     :PRINT#1,0
     :PRINT#1,133                                   ;;;STA $72
     :PRINT#1,114
     :PRINT#1,169                                   ;;;LDA #$A
     :PRINT#1,10

59700 PRINT#1,133                                   ;;;STA $71  (set up string pointer to beginning of input)
     :PRINT#1,113
     :PRINT#1,202                                   ;;;DEX
     :PRINT#1,138                                   ;;;TXA    (transfer number of characters in string to A)
     :PRINT#1,32                                    ;;;JSR $D68D   (convert string to floating-point in primary accum.)
     :PRINT#1,141

59760 PRINT#1,214
     :PRINT#1,162                                   ;;;LDX #ZL
     :PRINT#1,ZL
     :PRINT#1,160                                   ;;;LDY #ZH   (point to target location to store result)
     :PRINT#1,ZH

59780 PRINT#1,32                                    ;;;JSR $DAA6   (store primary memory contents in target)
     :PRINT#1,166
     :PRINT#1,218
     :PC=PC+45                                      ;;;count bytes (a lot!!)
     :GOTO 59500                                    ;;;and loop

60000 PRINT "FIRST PASS FINISHED!!!"                ;;;compiling program finished
     :PRINT"PROGRAM OCCUPIES";OC;"THROUGH";PC-1     ;;;print statistics

60020 PRINT"VARIABLES OCCUPY";OV;"THROUGH";OV+129

60060 PRINT#1,-1                                    ;;;mark end of first pass on tape file
     :IF GC=0 GOTO 61000                            ;;;skip if no jumps to be corrected

60080 FOR X=0 TO GC-1
     :PRINT#1,GOX(X,0)+32767                        ;;;print byte to be fixed
     :L=GOX(X,1)                                    ;;;look up desired line number to branch to
     :Z=LNX(L)+32767                                ;;;find what the program counter was there

60100 IF Z=0 THEN PRINT "TRANSFER TO NONEXISTENT LINE #";L
     :GOTO 61000                                    ;;;catch errors

60120 ZH=INT(Z/256)
     :ZL=Z-256*ZH
     :PRINT#1,ZL                                    ;;;record correct jump (JMP or JSR) target
     :PRINT#1,ZH
     :NEXT X                                        ;;;for all that need fixing

61000 PRINT#1,-1                                    ;;;mark end of file
     :CLOSE 1                                       ;;;that's all, folks!!!!!
```

Text continued from page 198:

byte value 137, COS is stored as 190, etc. This encoding system saves memory within BASIC programs; when a line is listed to the display, it is decoded so that it can be read in its original form. Encoding also makes the job of the FLOPTRAN compiler easier. Because BASIC encodes reserved words as 1-byte values rather than a sequence of letters, there is, for example, no confusion as to whether the letter G is a variable name or the first letter of a GOTO or GOSUB statement.

Compiler Program Analysis

Because the FLOPTRAN compiler is rather tiny, the language in its present version is a restricted subset of BASIC. The following paragraphs will describe each type of statement that FLOPTRAN understands and explain how it works. The line numbers refer to the statements in the BASIC program of listing 1 that process a given type of statement.

The LET Statement

Lines 58000 thru 58360: To assign a value to a variable during compilation, most BASIC systems allow us to use a statement like:

X=1.234E−5

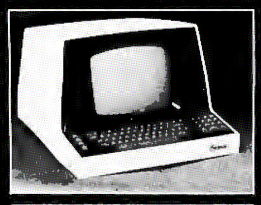
In most BASICs, however, it is also legal to say:

LET X=1.234E−5

In fact, in the original BASIC language, the use of the word LET was compulsory.

FLOPTRAN demands the presence of the reserved word LET. When the compiler sees LET, it notes the immediately following variable name, which will receive the assigned value on the right side of the equals sign. The compiler then literally copies the right-hand side of the equation into the blank space available in the subroutine

Why Your Next CRT Should Come From MICROMAIL:



SOROC IQ 120

- Displays 80 x 24, upper/lower case.
- Separate numeric keypad and cursor keys.
- Protected fields displayed at reduced intensity.

$689.00

SOROC IQ 140

- 117-key detachable keyboard with numeric cluster and cursor control.
- Insert/delete line, insert/delete character.
- Underline, blink, reverse, 1/2 intensity, protected and blank fields.
- Printer port with independent baud rate — prints line, partial or full screen.

$1099.00

TELEVIDEO 912/920

- Insert/delete line, insert/delete character, line/page erase.
- Reverse video, blinking, underline, ½ intensity, protected field, blank security field.
- Uses 7 x 10 dot matrix for a high quality u/l case display with descenders.
- Standard typewriter or teletype keyboard; numeric keypad.
- Model 920 includes 17 dedicated keys for function and editing.
- Block or character transmission, auxiliary printer port.
- Cursor up, down, left, right, return, home, load, read, tab and back tab.

Call for Low Price

TEC 510

- Reverse video, blinking, underline, 1/2 intensity, protected fields, blank security field.
- Transmit character, line, partial page, page, or unprotected data.
- Cursor up, down, left, right, return, home, plus load and read.

$699.00

We Also Represent the Following Manufacturers:

DIABLO DEC TEXAS INSTRUMENTS
ANADEX GTC TELETYPE

Write or Call In for Our Free Catalogue!

MICROMAIL

MICROMAIL • BOX 3297 • SANTA ANA, CA 92703
(714) 731-4338

TO ORDER: Send check or money order to: **MICROMAIL, P.O. Box 3297, Santa Ana, CA 92703.** Personal or company checks require two weeks to clear. Terminals in stock are shipped the business day after receipt of certified funds. All equipment includes factory warranty.

SHIPPING: We ship freight collect by UPS when possible. Larger terminals are shipped by motor freight. Air and express delivery is available on all products.

HANDLING: All orders are subject to MICROMAIL's handling charges. Less than $750.00, add 3%. $750.00 to $2,000.00, add 2%. Over $2,000.00 add 1%.

at line 49000, and calls that subroutine to evaluate the expression. Thus, statements such as:

LET P=17*LOG(π/137E8)+ATN(−2)

are perfectly legal ways to initialize a variable, as long as the right-hand side will fit into line 49000.

A LET assignment cannot, however, use variable names in its equation unless those variables have been defined at compilation time; it would *not* be legal to say:

LET A=−44:LET B=A

since A will take on its value of −44 only when the program is executed. (In this case, unless A were defined before running the compiler, B would be assigned the value 0.) The use of variables on the right-hand side of the assignment is covered later, under arithmetic operations.

After evaluating the expression that is to be assigned to some variable, the compiler writes out code (to device 1, the PET's cassette tape drive) which, in 6502 machine language, instructs the microprocessor to skip 5 bytes. Those 5 bytes are filled with the floating-point number to be assigned to the variable; then the compiler executes lines 58300 thru 58360, which takes the stored number and transfers it to the proper target location.

FLOPTRAN allows only single-letter variable names, A thru Z, and reserves storage for each variable, whether it is used or not. This simplifies storage of variables considerably and allows easy and instant access to any variable. The typical BASIC interpreter must scan through a table of variables to find the one it wants to access; the interpreter necessarily wastes a lot of time doing so.

The PRINT Statement

Lines 59000 thru 59400: To get numeric values to the display, statements of the form:

PRINT A

or

PRINT X,Y,Z

are used in FLOPTRAN. When multiple variables are printed by a single statement, they may be separated by either commas or semicolons. In both instances, the numbers that have been requested are translated from floating-point binary radix to ASCII-encoded decimal radix and are printed with a space between them. After the final variable is printed (or if no variables are to be printed) a carriage return and line feed (CR,LF) are printed to the screen.

The PET read-only-memory subroutine at location hexadecimal DCAF does the hard work of converting from binary to ASCII (American Standard Code for Information Interchange) decimal once the value to be converted is placed in the "floating-point accumulator" on memory page 0 (hexadecimal addresses 00B0 thru 00B5); the ASCII string to be displayed is left at the top of page one and ends with a byte of 00. The 6502 code that the compiler generates to take this string and print it out (using the subroutine at hexadecimal address FFD2) is documented in the comments with the FLOPTRAN compiler listing (listing 1).

Circle 138 on Inquiry card.

More Printing Terminals From MICROMAIL . . .

DIABLO 1650

- Prints at 40 cps, using 88, 92, or 96 char. metalized printwheels.
- Vertical resolution 1/48"; Horizontal 1/120". Capable of proportional spacing, bidirectional printing, and graphics under software control.
- Bidirectional normal and direct tabs. Left, right, top and bottom margins.

R.O. $2890.00
KSR $3285.00

DEC LA 34

(Shown with optional forms tractor and numeric keypad).

- Prints 10, 12, 13.2, or 16.5 characters per inch, upper/lower case.
- 2, 3, 4, 6, 8, or 12 lines per inch.
- Friction feed, paper width to 15 inches.

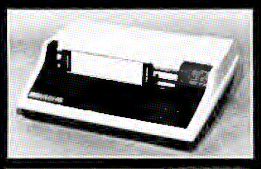
$969.00

Options:
— Numeric keypad — $80.00
— Adjustable forms tractor — $130.00

DIABLO 1640

- Uses plastic printwheel and prints at 45 cps. Otherwise, shares identical features with 1650 including:
  — Friction or tractor feed, up to 15" wide.
  — Cartridge ribbon, fabric or carbon.

R.O. $2745.00
KSR $3140.00



T.I. 810

- Includes upper/lower case option.
- Bidirectional printing at 150 cps.
- Tractor-feed forms, 3" to 15" wide.

$1599.00

Options:
— Forms length control — $100.00
— Vertical Format Control with Compressed Print — $125.00

ANADEX DP-9500/9501

- High Density Graphics
- Parallel, RS-232C, and Current Loop interfaces standard.
- Double width printing
- 132/175 or 132/220 columns.
- 50 to 200+ lines/min., 150/200 CPS 9 x 7/7 x 9 font or 120/200 CPS with 11 x 9/7 x 9 font.
- 9-wire print head, 650 million character life.
- Bi-Directional printing with shortest distance sensing logic.
- Adjustable width tractor paper feed.
- Complete forms control.

Call for Low Price

We Also Represent the Following Manufacturers:

TELETYPE GTC TeleVideo TEC SOROC

Write or Call In for Our Free Catalogue!

MICROMAIL
MICROMAIL • BOX 3297 • SANTA ANA, CA 92703
(714) 731-4338

TO ORDER: Send check or money order to: MICROMAIL, P.O. Box 3297, Santa Ana, CA 92703. Personal or company checks require two weeks to clear. Terminals in stock are shipped the business day after receipt of certified funds. All equipment includes factory warranty.

SHIPPING: We ship freight collect by UPS when possible. Larger terminals are shipped by motor freight. Air and express delivery is available on all products.

HANDLING: All orders are subject to MICROMAIL's handling charges. Less than $750.00, add 3%. $750.00 to $2,000.00, add 2%. Over $2,000.00 add 1%.

The INPUT Statement

Lines 59500 thru 59780: To read in numbers from the keyboard during program execution, statements of the form:

```
INPUT P
```

or

```
INPUT Q;R,S
```

are used. As with the PRINT statement, either commas or semicolons are valid delimiters to separate several variable names. (In fact, you can leave out the commas or semicolons and FLOPTRAN will not mind; but then BASIC will misunderstand you.)

The compiler generates code that prompts the user by printing out the variable name awaiting input, followed by a question mark. It then accepts input via the standard subroutine (hexadecimal location FFCF) and puts the input string into the BASIC input buffer on page 0. The read-only-memory subroutine at hexadecimal location D68D converts this string to a floating-point number, which is then stored in its proper location.

The IF Statement

Lines 58500 thru 58840: The only allowed forms of IF statement in FLOPTRAN-IV are the simplest ones:

```
IF W GOTO 252
```

and

```
IF J GOSUB 30
```

These statements test the variable whose name follows the IF; if that variable's value is nonzero, the statement is considered to be true, and the GOTO or GOSUB is executed. If the value of the variable is exactly 0, the statement is false, and the GOTO or GOSUB is skipped. Program execution proceeds with the next *statement*; note that, in the event of multiple statements per line, FLOPTRAN does *not* skip the rest of the line—unlike Microsoft BASIC, which executes the rest of the line only when the IF statement is true. (It is wise, therefore, to avoid using statements following an IF on the same line, to avoid incompatibility with BASIC.)

After testing for zero versus nonzero, this routine can do one of two things: if the expression is true (nonzero), control transfers to the appropriate GOTO or GOSUB routine; otherwise, control transfers to the next statement.

GOTO or GOSUB Statements

Lines 58700 thru 58840: Both of these statements are handled in identical ways; the only difference is whether a 6502 JMP (jump) instruction or a JSR (jump-to-subroutine) instruction is compiled to the tape file. Since the compiler, during its first scan through the FLOPTRAN source program, cannot know the correct absolute address to go to, it temporarily fills the 2 bytes following the JMP or JSR with 0s. It also records, in column 0 of array GO%, the location of those temporary filler 0s, so that the missing addresses can be added later. Since integer arrays use much less space than do floating-

Circle 140 on inquiry card.

point arrays in PET BASIC, GO% is an integer-type two-dimensional array. Inasmuch as integers must be within the range −32,767 to +32,767, FLOPTRAN subtracts 32,767 from the actual address before storing it and adds the same value when reading the address out (in line 60080).

Column 1 of GO% stores the target line number for the GOTO or GOSUB. After the compilation is finished, the target line numbers will be looked up in table LN% (lines 60000 thru 61000), which contains the actual address of the compiled code corresponding to the desired line number. Then, the address of the first of the pair of bytes to be replaced is written to tape, followed by the 2 correct bytes for the branch. When the contents of the tape are loaded to memory, these bytes have the effect of correctly filling in the previously incomplete addresses in the body of the compiled program.

Because the original PET read-only-memory routines limit arrays to 256 elements, FLOPTRAN-IV is set up to handle programs with line numbers in the range 0 thru 255 only. This restriction, however, can be removed with additional programming if more lines are needed.

The REM Statement

Lines 56500 thru 56540: Remarks, preceded by the reserved word REM, are not compiled; the compiler skips over them to the beginning of the next line of the program, just as is done in interpreted BASIC.

The RETURN Statement

Line 50420: This statement simply causes a 6502 RTS (return-from-subroutine) instruction to be written to tape.

Arithmetic Operations

Arithmetic operations with variables comprise the remainder of the FLOPTRAN-IV compiler's repertoire. Because this is such a tiny compiler, it does *not* handle multiple arithmetic operations per assignment. The program to be compiled, therefore, must have its arithmetic broken up into elementary operations. For example, a statement like:

```
D=(A−B)/LOG(C)
```

must be written out as:

```
D=A−B:E=LOG(C):D=D/E
```

(assuming that E is not being used otherwise).

Because of this restriction on the permissible forms of arithmetic statements, the part of the compiler which handles them is simple and consists only of lines 50440 thru 51120. One special type of equation, which is of the form:

```
V=W
```

is so simple that it is handled separately in lines 55600 thru 55660. The extensive comments included with the listing make this part of the compiler rather straightforward.

This completes our description of the FLOPTRAN-IV compiler program as given in listing 1. The compiler checks for most errors as it works through the source pro-

A REFURBISHED "SELECTRIC" ASCII TERMINAL
FOR THE SMALL BUSINESSMAN OR SERIOUS HOBBYIST.

The AJ 841 I/O terminal. Now available from dealers nationwide.

Demand for our AJ 841 I/O computer terminal has been great. And now it's getting even greater. So call your local computer shop dealer right away. Supply is limited! You may never have another opportunity like this one to buy your own *professional* terminal.

The AJ 841 features:

- Choice of serial RS 232 or parallel interface
- ASCII code
- 14.9 cps printout
- High quality Selectric printing
- Heavy-duty Selectric mechanism
- Off-line use as typewriter
- Documentation included
- 30-day warranty on parts and labor (details available on request)

Call toll-free now

For location of your nearest AJ dealer, call toll-free:

800/538-9721

California residents call 408/263-8520.

ANDERSON JACOBSON

Circle 142 on inquiry card.

BUSINESS - PROFESSIONAL - GAME SOFTWARE FOR APPLE AND TRS-80

☐ **HOME FINANCE PAK I: Complete package $49.95** Apple, TRS-80

☐ BUDGET: The heart of a comprehensive home finance system. Allows user to define up to 20 budget items. Actual expense input can be by keyboard or by automatic reading of CHECKBOOK II files. Costs are automatically sorted and compared with budget. BUDGET produces both monthly actual/budget/variance report and a year-to-date by month summary of actual costs. Color graphics display of expenses. . . $24.95

☐ CHECKBOOK II: This extensive program keeps complete records of each check/deposit. Unique check entry system allows user to set up common check purpose and recipient categories. Upon entry you select from this pre-defined menus to minimize keying in a lot of data. Unique names can also be stored for completeness. Rapid access to check files. Check register display scrolls for ease of review. 40 column printout. Up to 100 checks per month storage. Files accessible by BUDGET program $19.95

☐ SAVINGS: Allows user to keep track of deposits/withdrawals for up to 10 savings accounts. Complete records shown via screen or 40 column printer. $14.95

☐ CREDIT CARD: Keep control of your cards with this program. Organizes, stores and displays purchases, payments and service charges. Screen or 40 column printer display. Up to 10 separate cards $14.95

☐ **THE UNIVERSAL COMPUTING MACHINE: $39.95** Apple, TRS-80

A user programmable computing system structured around a 20 row x 20 column table. User defines row and column names and equations forming a unique computing machine. Table elements can be multiplied, divided, subtracted or added to any other element. User can define repeated functions common to a row or column greatly simplifying table setup. Hundreds of unique computing machines can be defined, used, stored and recalled, with or without old data, for later use. Excellent for sales forecasts, engineering design analysis, budgets, inventory lists, income statements, production planning, project cost estimates-in short for any planning, analysis or reporting problem that can be solved with a table. Unique cursor commands allow you to move to any element, change its value and immediately see the effect on other table values. Entire table can be printed by machine pages (user-defined 3-5 columns) on a 40 column printer. Transform your computer into a UNIVERSAL COMPUTING MACHINE.

☐ **COLOR CALENDAR:** HI-RES color graphics display of your personal calendar. Automatic multiple entry of repetitive events. Review at a glance important dates, appointments, anniversaries, birthdays, action dates, etc. over a 5 year period. Graphic calendar marks dates. Printer and screen display a summary report by month of your full text describing each day's action item or event. Ideal for anyone with a busy calendar. . (Apple Only) $19.95

☐ **BUSINESS SOFTWARE SERIES: Entire package $199.95** Apple, TRS-80

☐ MICROACCOUNTANT: The ideal system for the small cash business. Based on classic T-accounts and double-entry bookkeeping, this efficient program records and produces reports on account balances, general ledger journals, revenue and expenses. Screen or 40 column printer reports. Handles up to 500 journal entries per period, up to 100 accounts. Instructions include a short primer in Financial Accounting. $49.95

☐ UNIVERSAL BUSINESS MACHINE: This program is designed to SIMPLIFY and SAVE TIME for the *serious businessman* who must periodically Analyze, Plan and Estimate. The program was created using our Universal Computing Machine and it is programmed to provide the following planning and forecasting tools.

CASH FLOW ANALYSIS — PROFORMA BALANCE SHEET — SOURCE AND USE OF FUNDS
PROFORMA PROFIT & LOSS — SALES FORECASTER — JOB COST ESTIMATOR

Price, including documentation and a copy of the base program. Universal Computing Machine. . . . $39.95

☐ INVOICE: Throw away your pens. Use the ELECTRONIC INVOICE facsimile displayed on your CRT. The program prompts and you fill in the data. Includes 3 address fields (yours, Bill to and Ship to). Invoice No., Account No., Order No., Salesman, Terms, Ship Code, FOB Pt. and Date. Up to 10 items per sheet with these descriptions: Item No., No. of units, Unit Price, Product Code, Product Description, Total Dollar amount per item and invoice total dollar amount. Generates, at your option, hard copy invoices, shipping memos, mailing labels, audit copies and disc updates to master A/R files. (48K) $49.95

☐ BUSINESS CHECK REGISTER: Expanded version of the Checkbook II program. Handles up to 500 checks per month with complete record keeping. (48K). $29.95

☐ BUSINESS BUDGET: As described above and companion program to Business Check Register. Handles 500 transactions per month, up to 20 cost categories. Accesses BCR files for actual costs. (48K) . . . $29.95

☐ **ELECTRICAL ENGINEERING SERIES: Both programs $159.95** Apple

☐ LOGIC SIMULATOR: SAVE TIME AND MONEY. Simulate your digital logic circuits before you build them. CMOS, TTL, or whatever, if it's digital logic, this program can handle it. The program is an interactive, menu driven, full-fledged logic simulator capable of simulating the bit-time response of a logic network to user-specified input patterns. It will handle up to 1000 gates, including NANDS, NORS, INVERTERS, FLIP-FLOPS, SHIFT REGISTERS, COUNTERS and user-defined MACROS. Up to 40 user-defined, random, or binary input patterns. Simulation results displayed on CRT or printer. Accepts network descriptions from keyboard or from LOGIC DESIGNER for simulation. Specify 1000 gate version (48K required) or 500 gate version (32K required) $89.95

☐ LOGIC DESIGNER: Interactive HI-RES Graphics program for designing digital logic systems. A menu driven series of keyboard commands allows you to draw directly on the screen up to 15 different gate types, including 10 gate shape patterns supplied with the program and 5 reserved for user specification. Standard patterns supplied are NAND, NOR, INVERTER, EX-OR, T-FLOP, JK FLOP, D-FLOP, RS-FLOP, 4 Bit COUNTER and N-BIT SHIFT REGISTER. User interconnects gates just as you would normally draw using line graphics commands. Network descriptions for LOGIC SIMULATOR generated simultaneously with the CRT diagram being drawn. Drawing is done in pages of up to 20 gates. Up to 50 pages (10 per disc) can be drawn, saved and recalled. Specify 1000 gate (48K) or 500 gate (32K) system $89.95

☐ **MATHEMATICS SERIES: Complete Package $49.95** Apple only

☐ NUMERICAL ANALYSIS: HI-RES 2-Dimensional plot of any function. Automatic scaling. At your option, the program will plot the function, plot the INTEGRAL, plot the DERIVATIVE, determine the ROOTS, find the MAXIMA and MINIMA and list the INTEGRAL VALUE. For 16K $19.95

☐ MATRIX. A general purpose, menu driven program for determining the INVERSE and DETERMINANT of any matrix, as well as the SOLUTION to any set of SIMULTANEOUS LINEAR EQUATIONS. Disk I/O for data save. Specify 55 eqn. set (48K) or 35 eqn. (32K) $19.95

☐ 3-D SURFACE PLOTTER: Explore the ELEGANCE and BEAUTY of MATHEMATICS by creating HI-RES PLOTS of 3-dimensional surfaces from any 3-variable equation. Disc save and recall routines for plots. Menu driven to vary surface parameters. Games include BLACK HOLE gravitational curvature equations. . $19.95

☐ **ACTION ADVENTURE GAMES SERIES: Entire series $29.95** Apple only

☐ RED BARON. *Can you outfly the RED BARON?* This fast action game simulates a machine-gun DOG FIGHT between your WORLD WAR I BI-PLANE and the baron's. You can LOOP, DIVE, BANK or CLIMB in any one of 8 directions - and so can the BARON. In HI-RES graphics $14.95

☐ BATTLE OF MIDWAY: You are in command of the U.S.S. HORNETS' DIVE-BOMBER squadron. Your targets are the Aircraft carriers, Akagi, Soryu and Kaga. You must fly your way through ZEROS and AA FIRE to make your DIVE-BOMB run. In HI-RES graphics $14.95

☐ SUB ATTACK: It's April, 1943. The enemy convoy is headed for the CORAL SEA. Your sub, the MORAY, has just sighted the CARRIERS and BATTLESHIPS. Easy pickings. But watch out for the DESTROYERS - they're fast and deadly. In HI-RES graphics $14.95

☐ FREE CATALOG—All programs are supplied in disc and run on Apple II w/Disc & Applesoft ROM Card & TRS-80 Level II and require 32K RAM unless otherwise noted. Detailed instructions included. Orders shipped within 3 days. Card users include card number. Add $1.50 postage and handling with each order. *California residents add 6½% sales tax. Make checks payable to:*

VISA

SPECTRUM SOFTWARE

master charge

DEALER INQUIRIES INVITED — P.O. BOX 2084 · 142 CARLOW, SUNNYVALE, CA 94087 — FOR PHONE ORDERS · 408-736-4397

> FLOPTRAN does not allow the appearance of uncompiled constants in an arithmetic statement.

gram; when it finds an unrecognizable statement or a branch to a nonexistent line number, it closes the output file and displays the line number containing the erroneous statement number. This makes one part of the debugging task easy. FLOPTRAN's compatibility with BASIC makes preliminary program testing possible (though perhaps slow) using the PET's BASIC interpreter; this debugging option helps in the discovery and elimination of many other types of errors.

In abbreviated form, table 2 gives a summary of FLOPTRAN-IV syntax. Table 3 presents a very compact summary of the critical read-only-memory subroutine addresses in the original-style PET. (It is probably too compact to be useful to a novice user, but for the experienced assembly-language enthusiast, table 3 should make sense and be a valuable guide to the important entry points. The table is entirely in hexadecimal; P refers to the primary floating-point accumulator, where a 6-byte floating-point binary number is stored in hexadecimal locations 00B0 thru 00B5. S refers to the secondary accumulator, located at hexadecimal locations 00B8 thru 00BF; and A, X, and Y are internal registers of the 6502 microprocessor.) It is unnecessary, however, to understand table 3 or, indeed, to understand *anything* about 6502 machine-language programming to successfully use FLOPTRAN-IV. The compiler as given does all the necessary translation.

Sample FLOPTRAN-IV Program

Listing 2 is a sample program written entirely in the FLOPTRAN-IV language. It solves a simple problem from the Parker Brothers game *Risk*: if an attacker rolls three dice and a defender rolls two, what are the odds of each of the possible outcomes? This puzzle was analyzed and discussed in detail recently by Bruce D Barnett (see reference 1). If you don't play *Risk*, however, don't be concerned—the important thing for our purposes here is that this problem provides a good sample task to exercise the capabilities of the FLOPTRAN-IV compiler.

There are a few significant items to note in the FLOPTRAN-IV program of listing 2. First, since FLOPTRAN does not allow the appearance of uncompiled constants in an arithmetic statement, the first statement in line 3 could not be written as "A=A−1". Instead, it was necessary to define a variable name with the letter O (for "One") to be equal to one, as in the assignment "LET O=1" in line 1. Then the arithmetic statement to decrement A can be correctly written as "A=A−O" (line 3).

While the necessity to define constants in arithmetic statements is an occasional inconvenience, it gives the compiler a great speed advantage. In fact, one of the slowest and most wasteful operations in an interpreted BASIC program is the conversion of every written-out ASCII numeric constant into binary each time it is encountered. Because of this, it is a recommended practice in BASIC to define a variable for any constant that is repeatedly used, in order to avoid repeatedly converting the constant from ASCII-encoded decimal to binary. FLOPTRAN forces the user to follow this efficient procedure.

Circle 144 on Inquiry card.

Statement Type	Description	Examples
LET	initialize a variable during compilation; must be 60 characters or less, cannot use variables	LET A = 1 LET B = EXP(2)/2
PRINT	output a variable's value to screen; gives 1 space between values printed and a "return" at end	PRINT X PRINT P,Q;R PRINT
INPUT	read in a number from keyboard (prompts with variable name)	INPUT Z INPUT J;K,L
IF	conditional branch: the GOTO or GOSUB following will be executed if the variable after the IF is nonzero; skipped otherwise	IF S GOTO 0 IF T GOSUB 255
GOTO	unconditional transfer to line number	GOTO 89
GOSUB	subroutine call; returns to statement following GOSUB when done	GOSUB 176
REM	remarks; skip to next line number	REM THIS IS A TEST
RETURN	return from subroutine, and return to BASIC control at program end	RETURN
arithmetic	perform the specified floating-point arithmetic operation, statements can have only one operation to right of equals sign	C = D E = F + G H = − I J = ATN(K)

Notes:
1. All variables must have single-letter alphabetic names.
2. All lines must be numbered 0 thru 255.
3. Variables must be initialized before use; failure to do this results in a random initial value.

Table 2: *Summary of FLOPTRAN-IV syntax. Due to the nature of the compiler, the LET statement can be an expression of up to sixty characters but can use only arithmetic operators, functions, and constants. The arithmetic statement (which must not use the word LET) can use variables only and can contain, at most, one operator or function.*

A second item to note is the method used (in lines 25, 29, 33, 39, 43, 45, and 51) to compare the relative magnitudes of two variables. In line 25, for example, the real question being asked is: "Is X less than W?" Since FLOPTRAN-IV can only test for zero versus nonzero in its IF statement, the programmer must set up a test variable, Q in this case, that will answer the desired question by being either nonzero for *true* or zero for *false*. For example, after the assignment "Q=X−W" in line 25, the statement "Q=SGN(Q)" returns the value +1 if Q is positive, 0 if Q is zero, and −1 if Q is negative. By adding one in the statement "Q=Q+O", the ultimate result is that Q equals zero if and only if X is less than W. Within the framework of statements 25 and 27, this means that line 27 will be executed only when X is less than W.

Finally, note the "extra" RETURN statement at the end of the program (line 120). In BASIC, this statement is wrong and gives a RETURN WITHOUT GOSUB error message when the interpreter executes it; however, in FLOPTRAN-IV, the RETURN is necessary. It tells the microprocessor that the machine-language routine, which was called by either SYS or USR, is finished and that control now returns to the BASIC operating system. I neglected to include the RETURN in my first compilation of this sample program and was rather embarrassed when, after executing perfectly and printing out the correct answers, the machine "crashed" and had to be reset for further use.

Comparison of FLOPTRAN-IV and PET BASIC

A sophisticated optimizing compiler can usually generate extremely efficient machine-language programs; a common result is a program that runs ten to a hundred times faster than an equivalent interpreted version. FLOPTRAN doesn't do quite that well, although its performance is significantly better than that of PET BASIC in all my tests.

For the *Risk*-odds-calculation program of listing 2, for example, the program took 890 seconds (almost 15 minutes) to reach the answer using the BASIC interpreter. The FLOPTRAN-IV compiled version finished in only 97 seconds, which gives a speed advantage of better than 9 to 1! (Although it is possible that some speed could be gained by rewriting the BASIC program to use FOR-NEXT loops and other structures unavailable in FLOPTRAN-IV, I estimate it would be a small improvement, probably less than a factor of 2.)

The original BASIC program used as the FLOPTRAN source code occupies 627 bytes plus 126 bytes for variables, for a total of 753 bytes. The compiled FLOPTRAN program is bigger, taking 1437 bytes plus its standard 130-byte table of twenty-six variables (not all of which are used in this program), for a total of 1567 bytes. The compilation time for this program is about 4.5 minutes, most of which is spent writing to tape.

Another simple test program, one that doesn't show such a great speed advantage for the FLOPTRAN compiler, is a program to factor a large odd number by trying

3a

Hexadecimal Address	Function of Routine	Hexadecimal Address	Function of Routine
0000	USR(P)	D95E	[(A) + 100(Y)]→S, separating sign, set sign comparison, return with (B0) in A
CED6	S OR P	D9E4*	S / P
CED9	S AND P	DA74	[(A) + 100(Y)]→P, separating sign
D264	FRE(P)	DAA6	roundoff(P)→[(X) + 100(Y)], merging sign
D285	POS(P)	DACE	S→P
D349	STR$	DADE	P→S, with rounding
D5C4	CHR$	DAE1	P→S, without rounding
D5D8	LEFT$	DAED	roundoff(P)
D604	RIGHT$	DAFD	SGN(P)→A
D60F	MID$	DB0B	SGN(P)
D654	LEN	DB2A	ABS(P)
D663	ASC	DB9E	INT(P)
D685	VAL	DE24	SQR(P)
D6E6	PEEK	DE2E*	S 1 P
D728*	S − P	DEA0	EXP(P)
D73C	JSR D95E, then S + P	DF45	RND(P)
D73F*	S + P	DF9E	COS(P)
D7AC	normalize P	DFA5	SIN(P)
D8BF	LOG(P)	DFEE	TAN(P)
D8FD	JSR D95E, then S * P	E048	ATN(P)
D900*	S * P		

3b

Hexadecimal Address		Contents	Examples: Hexadecimal Values of Bytes for	
S pseudoregister	P pseudoregister		S = e = 2.71828...	P = π/2
B8	B0	exponent + 80	82	81
B9	B1	fraction, most significant byte	AD	C9
BA	B2	fraction, byte 2	F8	0F
BB	B3	fraction, byte 3	54	DA
BC	B4	fraction, least significant byte	59	A2
BD	B5	sign	00	FF
BE	—	sign comparison	FF	—
BF	—	roundoff byte	00	—

3c

Hexadecimal Address	Value stored in floating (merged sign) notation
CDBC	π
DDE3	1/2
E01A	π/2
E01F	2π
E024	1/4

Table 3: *Overview of useful addresses and constants used by the original-version PET. Table 3a is a list of entry points to subroutines within the PET's read-only-memory firmware. Table 3b explains the makeup of the two floating-point pseudoregisters P and S. Table 3c gives the locations of five constants stored in the read-only-memory area of the PET. In table 3a, the routines marked with an asterisk (ie: all arithmetic operations) must be set up before execution as follows: hexadecimal location 00BE must contain the exclusive-OR of the contents of hexadecimal locations 00B5 and 00BD. Also, the contents of hexadecimal location 00B0 must be placed in the 6502 accumulator.*

all odd numbers as potential divisors, from 3 up to the square root of the number. The compiled FLOPTRAN code takes 60 seconds to factor the number 567,890,123; the same program run in PET BASIC requires 201 seconds, a speed advantage of better than 3 to 1 for the FLOPTRAN compiler. But, after the BASIC program has been rewritten for speed, the revised program executes in 101 seconds, giving a speed advantage of 1.7 to 1 for compiled FLOPTRAN.

One of the reasons that the BASIC interpreter does not have much of a disadvantage in this factorization problem is that the BASIC program needs to use only a few variables, so that little time is wasted scanning through a variable table. Also, this program performs very little decimal-to-binary conversion and a small amount of branching from one line number to another (which requires more BASIC scanning to find the target line). These three factors contribute to this problem's lower

Listing 2: *Sample FLOPTRAN-IV program. This program calculates the possible outcomes of a dice roll in the game* Risk, *with the attacker rolling three dice and the defender rolling two dice. See text for details.*

```
LIST -120

  1 LET F=0
   :LET G=0
   :LET H=0
   :LET J=0
   :LET A=7
   :LET B=6
   :LET C=6
   :LET O=1

  3 A=A-O
   :IF A GOTO 9

  4 GOTO 100

  5 LET B=6

  7 LET A=6

  9 LET D=7
   :LET E=6

 11 D=D-O
   :IF D GOTO 21

 13 E=E-O
   :IF E GOTO 17

 15 GOTO 3

 17 LET D=6

 21 I=I+O

 23 V=A
   :W=B
   :X=C
   :Y=D
   :Z=E

 25 Q=X-W
   :Q=SGN(Q)
   :Q=Q+O
   :IF Q GOTO 29

 27 T=X
   :X=W
   :W=T

 29 Q=W-V
   :Q=SGN(Q)
   :Q=Q+O
   :IF Q GOTO 39

 31 T=W
   :W=V
   :V=T

 33 Q=X-W
   :Q=SGN(Q)
   :Q=Q+O
   :IF Q GOTO 39

 35 T=X
   :X=W
   :W=T

 39 Q=Z-Y
   :Q=SGN(Q)
   :Q=Q+O
   :IF Q GOTO 43

 41 T=Z
   :Z=Y
   :Y=T

 43 Q=Z-X
   :Q=SGN(Q)
   :Q=Q+O
   :IF Q GOTO 51

 45 Q=V-W
   :Q=SGN(Q)
   :Q=Q+O
   :IF Q GOTO 53

 47 GOTO 61

 51 Q=Y-W
   :Q=SGN(Q)
   :Q=Q+O
   :IF Q GOTO 71

 53 G=G+O
   :GOTO 11

 61 F=F+O
   :GOTO 11

 71 H=H+O
   :GOTO 11

100 B=B-O
   :IF B GOTO 7

110 C=C-O
   :IF C GOTO 5

120 PRINT F,G,H
   :F=F/I
   :G=G/I
   :H=H/I
   :PRINT F,G,H
   :RETURN
```

Notes on CHEEP PRINT

CHEEP PRINT is a software/hardware combination invented by Charles A McCarthy (1359 W Idaho Ave, St Paul MN 55708) to print the listings given in this article; it allows a user with a printer and modem to interface the two to a personal computer. Functionally, CHEEP PRINT produces a rapid sequence of 2025 and 2225 Hz tones (via a short machine-language program) that are converted into RS-232C ASCII character data by a standard Bell-103a-type modem. When storing programs, a simple low-pass filter is inserted between the computer and the cassette recorder to smooth the output waveform for the cassette recorder. A BASIC driver program formats the program listing and passes characters, one at a time, to the machine-language subroutine that generates the tones.

In this implementation, a Commodore PET microcomputer and a Teletype Model 43 are used. The machine-language subroutine for the PET is shortened by the use of the internal clock and programmable shift register of a 6522 VIA (Versatile Interface Adapter) chip within the PET; these are used to generate the required square-wave tones, which are made available on the CB2 pin of the PET's user port.

execution time. On the other hand, the *Risk* program uses many variables and branches, giving the FLOPTRAN compiled program more of an edge there.

Final Remarks

When the FLOPTRAN-IV compiler program is run, the result is a set of numbers that have been written to the PET's cassette tape drive. This tape, which contains the location and contents of the machine-language program that is the compiled equivalent of the given FLOPTRAN-IV source program, must be loaded into memory before the compiled program can be run. This is done by running the FLOPTRAN-IV loader program in listing 3.

This short BASIC program reads in the starting address for the compiled code (in the variable PC, which serves as a program counter), POKEs the following numbers into consecutive memory locations until it encounters a −1 end-of-program marker (written to the tape by line 60060 of listing 1), then reads in the addresses and correction-bytes to put into those GOTO and GOSUB instructions that need them. A final −1, written to tape by line 61000 of listing 1, ends the loading process.

The loader program prints out the starting and ending addresses of the compiled code so that the user can check that no data blocks were dropped by the PET's tape recorder. I did not include a subroutine in the FLOPTRAN compiler to force the PET's tape-recorder motor to leave extra gaps between data blocks (if necessary, this can be done with a POKE 59411,53 instruction). Instead, I have added a 5000 to 10,000 μF capacitor which is connected across the motor terminals during recording; this automatically causes the recorder to leave enough space. The discharging capacitor makes the motor run on for some time after it would normally stop. If your machine needs longer gaps, you may add the above POKE statement, perhaps between lines 49100 and 49120 (this is within the frequently called subroutine which fetches the next nonblank character).

By choosing the starting address for the compiled program appropriately, you can load the compiled program into memory and then load a monitor program that will allow you to save the compiled code as a binary machine-language file. Generally these binary files are much more reliable than ASCII data files, and of course, they load much faster.

You can also locate a compiled program in the video-display area of memory, locations decimal 32,768 thru 33,791, provided that the program is small enough to fit. Running such a program gives very interesting visual effects.

SORCERER* SOFTWARE!

Unless otherwise noted, all programs are on cassette and require only 8K of memory.

FORTH

new! Now Sorcerer owners can enjoy the convenience and speed of the fascinating FORTH programming language. Based on FIGFORTH and written by James Albanese, this version was designed especially for the Sorcerer and includes the capability to read and write data (screens) to cassette tape and a complete on-screen editor. Requires at least 16K of RAM. **$49.95**

new! GRAPHICS ANIMATION by Lee Anders. This package provides the BASIC programmer with a powerful set of commands for graphics and animation. The program is written in machine language but is loaded together with your BASIC program and graphics definitions with a CLOAD command. Any image from a character to a large graphic shape may be plotted, moved, or erased with simple BASIC commands. Encounters of plotted character sets with background characters are detected and background images are preserved. Contains a medium resolution plotting routine. A keyboard routine detects key presses without carriage returns. Includes a separate program for constructing images. **$29.95**

new! STARBASE HYPERION™ by Don Ursem. At last, a true strategic space game for the Sorcerer! Defend a front-line Star Fortress against invasion forces of an alien empire. You create, deploy, and command entire ship squadrons as well as ground defenses in this complex tactical simulation of war in the far future. Written in BASIC and Z-80 code. Full graphics and realtime combat status display. Includes full instructions and STARCOM battle manual. Requires at least 16K of RAM. **$17.95**

new! HEAD-ON COLLISION™ by Lee Anders. You are driving clockwise and a computer-controlled car is driving counter clockwise. The computer's car is trying to hit you head on, but you can avoid a collision by changing lanes and adjusting your speed. At the same time you try to drive over dots and diamonds to score points. Three levels of play, machine language programming, and excellent graphics make this game challenging and exciting for all. At least 16K of RAM is required. **$14.95**

new! LUNAR MISSION by Lee Anders. Land your spacecraft softly on the moon by controlling your craft's three propulsion engines. Avoid lunar craters and use your limited fuel sparingly. You can see both a profile view of the spacecraft coming down and a plan view of the landing area. Land successfully and you get to view an animated walk on the moon. Nine levels of play provide a stiff challenge to the most skillful astronaut. Requires at least 16K of RAM. **$14.95**

new! HANGMAN/MASTERMIND by Charles Finch. Two traditional games are brought to life by Sorcerer graphics. HANGMAN has three different vocabulary levels for you to choose from. In MASTERMIND, the computer selects a four-character code and you have to uncover it. These two games provide an enjoyable way for young people to develop their vocabulary and their logical reasoning ability. Written in BASIC. **$11.95**

QS SMART TERMINAL by Bob Pierce. Convert your Sorcerer to a smart terminal. Used with a modem, this program provides the capability for you to communicate efficiently and save connect time with larger computers and other microcomputers. The program formats incoming data from time-sharing systems such as The Source for the Sorcerer video. Incoming data can be stored (downloaded) into a file in RAM. Files, including programs, may be saved to or loaded from cassette, listed on the video, transmitted out through your modem, or edited with an on-board text editor. Interfaces with BASIC and the Word Processor Pac. **$49.95**

DPX™ (Development Pac Extension) by Don Ursem. Serious Z80 program developers will find this utility program to be invaluable. Move the line pointer upward. Locate a word or symbol. Change a character string wherever it occurs. Simple commands allow you to jump directly from EDIT to MONITOR or DDT80 modes and automatically set up the I/O you want for listings. Built-in serial driver. Stop and restart listings. Abort assembly with the ESC key. Save backup files on tape at 1200 baud. Load and merge files from tape by file name. Versions for 8K, 16K, 32K, and 48K Sorcerer all on one cassette. Requires the Sorcerer's Development Pac. **$29.95**

Other utility programs:

PLOT by Vic Tolomei. High res and low res modes	**$14.95**
SHAPE MAKER™ by Don Ursem. An on-screen character maker	**$14.95**
DEBUG by Bob Pierce. Debug machine language programs	**$14.95**
SOFTWARE INTERNALS MANUAL by Vic Tolomei. A 64-page book	**$14.95**

Other game programs:

MARTIAN INVADERS™ by James Albanese	**$14.95**
NIKE II™ by Charles Finch and Bob Broffel	**$11.95**
TANK TRAP by Don Ursem	**$11.95**
MAGIC MAZE™ by Vic Tolomei	**$11.95**
FASTGAMMON™ by Bob Christiansen	**$19.95**

QS QUALITY SOFTWARE
6660 Reseda Blvd., Suite 105, Reseda, CA 91335
Telephone 24 hours, seven days a week: (213) 344-6599

WHERE TO GET IT: Ask your nearest Sorcerer dealer to see *Quality Software's* Sorcerer programs. Or, *if you prefer*, you may order directly from us. MasterCharge and Visa cardholders may telephone their orders and we will deduct $1 from orders over $19 to compensate for phone charges. Or mail your order to the address above. California residents add 6% sales tax. *Shipping Charges:* Within North America orders must include $1.50 for first class shipping and handling. Outside North America the charge for airmail shipping and handling is $5.00 — payable in U.S. currency.

*The name "SORCERER" has been trademarked by Exidy, Inc.

Listing 3: *FLOPTRAN-IV loader program. This program is executed to transfer the compiled FLOPTRAN-IV program from the PET cassette tape drive into memory. See the "How to Use the FLOPTRAN-IV Compiler" text box for details.*

```
10 OPEN 1: INPUT #1, PC: PRINT "BEGIN AT"; PC
20 INPUT #1, A: IF A=>0 THEN POKE PC,A:
      PC=PC+1: GOTO 20
30 PRINT "END AT"; PC-1
40 INPUT #1, PC: IF PC<0 THEN CLOSE 1: END
50 INPUT #1,A: POKE PC,A: INPUT #1,A:
      POKE PC+1,A: GOTO 40
```

What other logical improvements can be made to make FLOPTRAN faster and more useful? String-handling capabilities sometimes come in handy; it would not be hard to allow the compiled program to recognize string variables and print them when requested, though I'm not sure how you would go about manipulating strings.

Perhaps the compiler should not allocate a fixed amount of space to a table of only twenty-six variables; instead, it might define variables when needed by the program. Arrays of variables, which are also valuable to have, should not be hard to implement.

FLOPTRAN suffers from the simplicity of its conditional-branching structure; an IF statement that could test for conditions other than zero/nonzero would be a profitable addition. It also would be good to have a looping structure available, similar to BASIC's FOR...NEXT, or perhaps like FORTRAN's DO-loop. Even a simple decrement-and-skip-on-zero calculator-type instruction would save time and space in many programming situations. Simple integer variables and integer arithmetic might also be worth having for many tasks.

I've resisted pursuing some of the above modifications to FLOPTRAN-IV in order to retain the language's compatibility with BASIC; perhaps that compatibility should be given up in exchange for more power and efficiency.

I would be interested to hear from readers who have used or improved upon FLOPTRAN or who have adapted it to other machines. The idea of a tiny compiler that uses an existing BASIC interpreter (especially one that permanently resides in read-only memory) is a simple idea, but I haven't seen it discussed in print. It is possible that if a manufacturer were to have a few hundred to a couple of thousand bytes of read-only memory available beyond the BASIC interpreter, he could put a better version of a FLOPTRAN-like compiler in that space so that it would always be available. Interpreted BASIC is wonderful, but there are some programming jobs that need more speed and efficiency — jobs only a compiled language can fill. ■

Reference

Barnett, Bruce D, "How to Program a Complex Problem," *Personal Computing*, September 1979, pages 26 to 30.

Acknowledgments

I would like to thank Jim Butterfield and Charles A McCarthy for helpful ideas and programming tools, without which FLOPTRAN would have taken far longer to develop.

Circle 149 on inquiry card.