Editor's Note

We are particularly pleased to include this article by Dick and Jill Miller in this FORTH theme issue. One of the problems with past BYTE language issues has been the lack of concrete examples of the language being showcased—namely, a full, nontrivial program that does something useful or fun and, at the same time, shows an example of the language at its best.

The program BREAKFORTH, written for the MMSFORTH language running on the Radio Shack TRS-80, does show the language FORTH at its best. This real-time video game, which is a version of the arcade-type game that requires the user to chip away at a "brick wall" by directing a bouncing ball at it with a paddle, is what Dick Miller calls "electronic flypaper"—a game so addictive that it keeps people trapped at their TRS-80, unable to stop playing.

In addition to being playable (quite a testament to the speed of FORTH, especially if you have ever seen the same game written in TRS-80 BASIC), the game also gives an example of how a good FORTH program is put together, as well as how it can be more readable when properly written out with adequate indentation and comments.

Another departure from previous language issues is the availability of the language FORTH at reasonable cost on a wide range of microcomputers (see chart of FORTH sources, elsewhere in this issue). Miller Microcomputer Services (MMS) supplies one of the most complete and well-supported versions of FORTH available, along with a newsletter and other FORTH products available at reasonable prices. (For example, MMS sells a FORTH software package that adds floating-point arithmetic (both single- and double-precision), complex arithmetic, and a full Z80 assembler, all on floppy disk for $29.95.)

This article was produced with the help of two other people not yet mentioned. The first is Tom Dowling, who wrote the MMSFORTH language for the TRS-80 and who does a large portion of the FORTH programming for MMS. The second person is Arnold Schaeffer, who wrote the BREAKFORTH program as his first FORTH program. If this achievement were not impressive enough, then I should add that Arnold is a high school student. This is proof that FORTH can be learned by anyone with sufficient enthusiasm for the language.

Analyzing the BREAKFORTH program is a great way to learn about FORTH and how to program in it. The program can be typed in as is on a TRS-80 using MMSFORTH's full-screen editor and virtual memory, but I suggest that you first read John James' article in this issue, "What Is FORTH? A Tutorial Introduction," before seriously studying the BREAKFORTH program.

*One final note on alteration: this program is meant to work on a TRS-80 Model I running MMSFORTH. Users of other FORTH systems having a graphic display of 48 by 128 resolution or better can probably get the program running by rewriting some words unfamiliar to their system. Some information designed to help in this conversion effort has been supplied in this article....***GW**

BREAKFORTH Into FORTH!

A Richard Miller and Jill Miller
Miller Microcomputer Services
61 Lake Shore Rd
Natick MA 01760

About the Authors

A Richard (Dick) and Jill Miller founded Miller Microcomputer Services in 1977 as a consulting firm specializing in support for the Radio Shack TRS-80. After continued dissatisfaction with other languages available for the TRS-80 (FORTRAN, COBOL, Pascal, PILOT, BASIC), they settled on FORTH as a language that combines the seemingly incompatible traits of language complexity, high operating speed, and low memory overhead. They released their first version of MMSFORTH (version 1.5) in June 1979, and have been improving disk and cassette versions of the system ever since. MMSFORTH resembles the FORTH Inc version of the language called microFORTH, and was written independently with permission from that company.

Introduction to BREAKFORTH

This BREAKFORTH program was created by Arnold Schaeffer. The program, which was purchased by MMS, has received minor modifications and is now included with the purchase of MMSFORTH version 1.9 (on a different range of blocks from those shown here, blocks 69 thru 74). We think it is a classic game as is, and fully expect individuals to modify it in accord with their game preferences—for their individual use.

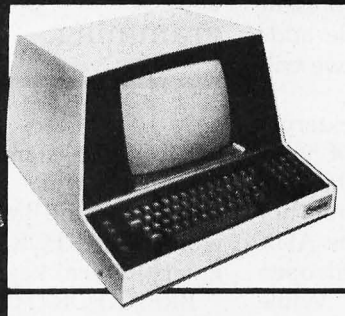
The BREAKFORTH program is a straightforward one, although it is not a trivial one. It combines many of the techniques of FORTH and can be followed easily with a little time and study. Figure 1 shows a typical BREAKFORTH video display, with an operator-controlled game paddle at the bottom, a bouncing ball, and a barrier to be knocked out one brick at a time by successive bounces until all the bricks have been cleared away. Each removed brick scores one point or more depending on its level, and there is a surprise bonus for a completely cleared barrier. Ball speed and number of balls are selectable, but be warned that, as you bounce your way up to the higher layers, the ball speed increases! You might want to start with short games using five balls and

NO FRILLS! NO GIMMICKS! JUST GREAT

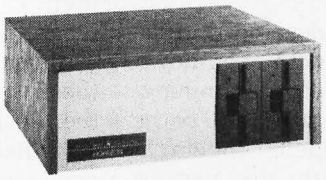
DISCOUNTS

MAIL ORDER ONLY

ATARI 800
Personal Computer System
$799.00

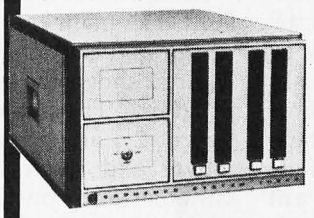


SOROC Technology
IQ 120 **$699.00**
IQ 140 **999.00**



NORTHSTAR

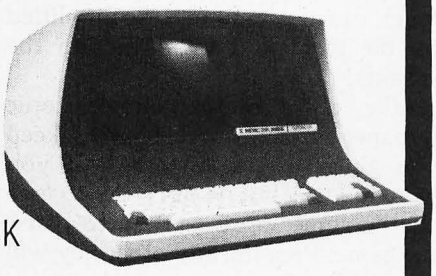
Horizon II 32K	**$2349.00**
Horizon Ii Quad	**2799.00**
Horizon II 64K	**2999.00**
Horizon Quad 64K	**3399.00**



CROMEMCO
System 3 **$5695.00**
Z2H **7995.00**

TELEVIDEO
912 **$749.00** | 920 **$799.00**

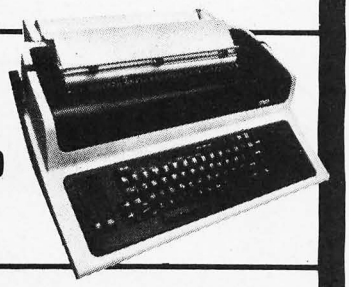
INTERTEC
Superbrain 32K Computer
$2495.00
Superbrain 64K
$2795.00

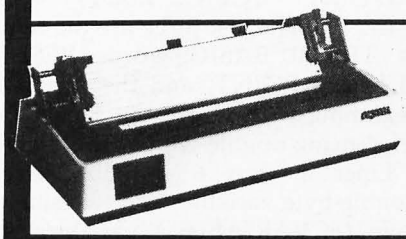


HAZELTINE

1420	**$795.00**
1500	**$849.00**
1510	**$1049.00**
1520	**$1229.00**

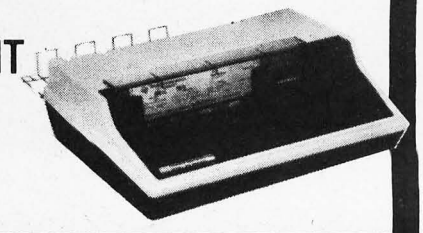
DECwriter IV
LA 34 **$979.00**





OKIDATA
Microline 80
$699.00

TEXAS INSTRUMENT
810 Multi Copy Impact Printer
$1499.00



We'll meet or beat any advertised prices!

Most items in stock for immediate delivery. Factory-sealed cartons.
Full manufacturer's guarantee.

DATA DISCOUNT CENTER

Box 100 135-53 Northern Blvd., Flushing, New York 11354

Visa • Master Charge • N.Y.S. residents add appropriate Sales Tax • Shipping F.O.B. N.Y.

PHONE ORDERS CALL
212
465-6609

a ball speed of seven. Fifty balls and a speed of four will present a challenge for high scorers.

BREAKFORTH offers some other features, too. As you and your friends try for better scores, a BEST score is kept to challenge your present effort. In addition, the paddle adds backspin in certain cases that we will leave you to discover.

To add sound, plug an external speaker into the EAR jack of your cassette tape recorder, attach the middle cable from the keyboard unit (not the motor remote cable) to the AUX jack of the tape recorder, and open the tape compartment door. While depressing the write-protect detector switch at the left side of the back of this compartment, simultaneously press the Record and Play keys. This procedure allows the cassette tape recorder to be used as an amplifier. The BREAKFORTH program manipulates the cassette port (normally used for writing a program to tape), causing a sound to be amplified by the recorder and played on the speaker.

Like other brands of electronic flypaper, BREAKFORTH may keep you glued to the keyboard. If you have to leave but do not want to give up the game, press shift-@ to pause the game. Pressing any other key will cause the game to resume where you left off. To start a new game in midstream while keeping the BEST score, press the Break key, type in the word BREAKFORTH , and press the Enter (Return) key.

> BREAKFORTH is developed in the FORTH manner, with top-down design and bottom-up programming.

BREAKFORTH is developed in the FORTH manner, with top-down design and bottom-up programming. Figure 2 shows the organization of the program. These modules shown in figure 2, along with the various 1-byte and single-precision (2-byte) variables and constants they invoke, are listed with explanations in table 1, a directory of the BREAKFORTH words that this program will add to the FORTH vocabulary.

The program's source code is on six consecutive blocks, and in this case happens to be located on blocks 50 thru 55; see listing 1. In MMSFORTH, one enters { 50 6 LOADS } to load the program—that is, to compile and execute all the information on blocks 50 thru 55, ending with the immediate execution of the word BREAKFORTH from line 15 of block 55 (which causes the program to be run). (Other versions of FORTH that lack the consecutive-blocks word, LOADS , will have another way of doing this.)

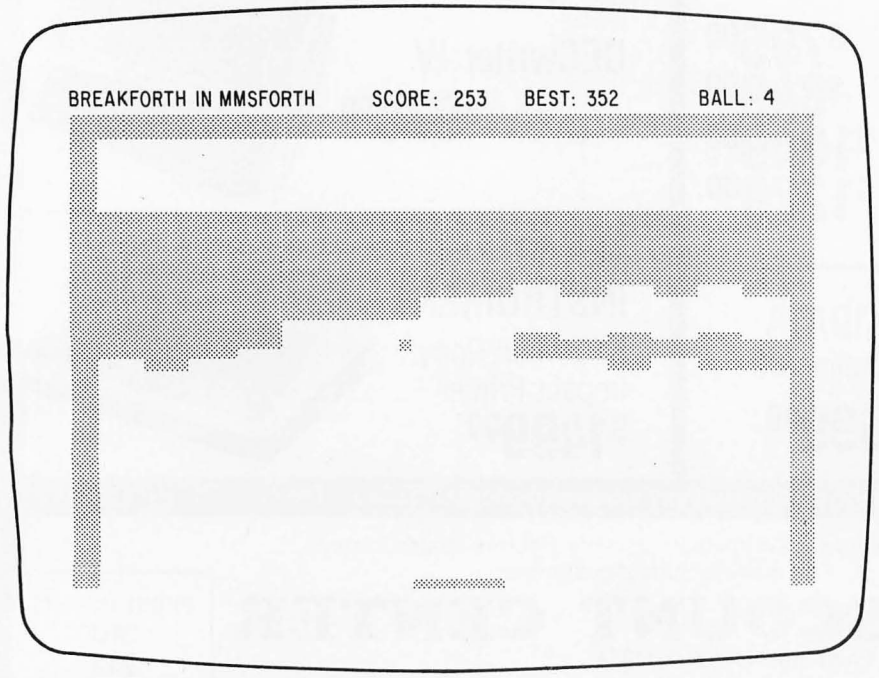


Figure 1: *One view of the TRS-80 video screen during a BREAKFORTH game.*

The First Block

Let us take a detailed look at block 50 in listing 1. Lines 0 thru 2 are all comment lines, as are any words surrounded by parentheses. Notice that because FORTH words are set off by spaces on either side, the "begin comment" word, { (} , must be separated from the first word of the comment by at least one space. (Because of the way { (} is defined, the closing parenthesis need not be separated from the last word of the comment by a space.)

Most definitions in FORTH begin with a colon ({ : }) and end with a semicolon ({ ; }), where the first word after the colon is the word being defined. In line 3, the first word defined is TASK . Since the only word following TASK is the closing semicolon, we can conclude that the word TASK does not do much. However, it does serve as a "bookmark," marking the beginning of the words and variables that are specific to this application (game). We will come back to TASK later, at the end of block 55.

Line 3 also causes two other blocks on the MMSFORTH system disk to be loaded into memory. Block 32, when loaded, adds several special-purpose words having to do with random numbers: RANDOMIZE and RND . Block 33, when loaded, adds several words that have to do with graphics: DCLR , DSET , { D? } , ECLR , ESET , and { E? } . (The last three are the same as TRS-80 BASIC words RESET, SET, and POINT, and the variables beginning with D are the same, but referencing double-width characters.)

Lines 4 thru 6 initialize seven double-byte variables and two single-byte (CVARIABLE) variables. In FORTH, unless specified, all variables, constants, and stack entries are 16 bits (2 bytes) long. See table 1 for the meaning of these variables.

Line 7 defines a new word, LINE , using a colon to begin the definition and a semicolon to end it. Several spaces (usually three) are placed be-

THE NEXT GENERATION OF MICROCOMPUTERS IS HERE
AT QUASAR DATA PRODUCTS

16 BIT POWER
Z-8000
AND STILL RUN YOUR 8 BIT SOFTWARE

IF YOU SEE IT OUR WAY THEN WE THINK WE HAVE THE PRODUCTS FOR YOU:

- THE S-100 BUS IS HERE TO STAY. IT IS NOT THE GREATEST BUT WITH PROPER TERMINATION IT WORKS RELIABLY AT HIGH SPEEDS, AND SINCE IT IS NOW AN IEEE STANDARD, IT IS WELL DEFINED.
- THE 8 BIT SYSTEMS ARE USEFUL BUT THEY ARE THE LIMITING FACTOR FOR MANY APPLICATIONS.
- THE 16 BIT SYSTEMS ARE THE WAY FUTURE SYSTEMS WILL GO. WHY NOT? THERE IS VERY LITTLE PRICE DIFFERENCE AND AN ORDER OF MAGNITUDE PERFORMANCE DIFFERENCE.
- THE REAL USEFULNESS OF THE 16 BIT MICROPROCESSORS WILL BE DETERMINED BY THE SOFTWARE.
- THE SYSTEMS USING 5¼ INCH DISK DRIVES REALLY DO NOT HAVE ADEQUATE MEMORY STORAGE OR COMPUTER POWER FOR MANY BUSINESS OR SCIENTIFIC APPLICATIONS.
- SIXTY-FOUR KILOBYTES OF ADDRESSABLE RAM, THE MAXIMUM FOR 8 BIT SYSTEMS, IS NOT ADEQUATE FOR MANY BUSINESS OR SCIENTIFIC APPLICATIONS.
- IT IS NOT WORTH BUYING 8 BIT SYSTEMS OR BOARDS NOW IF YOU CAN GET THE SAME SOFTWARE WITH 16 BIT SYSTEMS AT ABOUT THE SAME PRICE.
- THE NEW 16 BIT MICROPROCESSORS HAVE POWER COMPARABLE TO MINICOMPUTERS BUT DO NOT REQUIRE THE SAME OVERHEAD IN TERMS OF DOWNTIME, MAINTENANCE, OR INITIAL INVESTMENT THEY ARE MORE VERSATILE IN MANY APPLICATIONS SUCH AS REAL TIME APPLICATIONS.

THIS IS WHAT QDP HAS AVAILABLE:

- A Z-8000 BOARD THT CAN PLUG INTO YOUR EXISTING S-100 BUS SYSTEM (SEE BELOW FOR DESCRIPTION)
- A COMPLETE Z-8000 SYSTEM (SEE BELOW FOR DESCRIPTION)
- A Z-8000 SYSTEM CONFIGURED FOR YOUR EXACT NEEDS.
- SOFTWARE TO ALLOW YOU TO RUN ALL THE AVAILABLE Z-80/8080 SOFTWARE INCLUDING CP/M
- SOFTWARE THAT INCLUDES A MONITOR, DEBUGGER, DISASSEMBLER, AND BASIC
- SOFTWARE OPTIONS A) EXTENDED MONITOR B) PASCAL C) SIMULATORS FOR 8080, Z-80, 6800, 6502, 1802
- A Z-80 SYSTEM (QDP-100) THAT IS UPWARD COMPATIBLE WITH THE Z-8000

THIS IS WHAT IS COMING FROM QDP:

- A 256 KILOBYTE RAM CARD.
- UNIX[2] OPERATING SYSTEM.

Z-8000 SERIES 16 BIT CPU S-100 BOARD — CAN BE PLUGGED INTO YOUR EXISTING SYSTEM $695.00

INDUSTRIAL QUALITY

- FULLY S-100 IEEE COMPATIBLE
- SUPPORTS EXISTING 8 BIT MEMORY AND 8 BIT PERIPHERAL BOARDS
- CAPABLE OF READING AND/OR WRITING 8 BIT, 16 BIT OR MIXED 8 BIT AND 16 BIT MEMORIES AUTOMATICALLY
- 8 BIT AND/OR 16 BIT PERIPHERAL MODULES CAN SIMULTANEOUSLY CO-EXIST IN THE SAME BUS WITHOUT ANY MODIFICATIONS
- CAPABLE OF OPERATING AS A SLAVE PROCESSOR TO ENABLE YOUR EXISTING CPU TO CONTROL THE Z-8000
- SUPPORTS ON-BOARD HARDWARE SINGLE STEPPING
- SUPPORTS EITHER SEGMENTED CPU OR NON-SEGMENTED CPU
- POWER-ON AND RESET JUMP DIP SWITCH SELECTABLE
- JUMPER SELECTABLE 2 OR 4 MHz. OPERATION
- DIP SWITCH SELECTABLE NUMBER AND TYPE OF WAIT STATES
- SOFTWARE
  - Z-80 EMULATOR ENABLES YOU TO EXECUTE YOUR EXISTING 8 BIT SOFTWARE **WITHOUT** ANY MODIFICATIONS AND ALLOWS YOU TO RUN CP/M IMMEDIATELY
  - EXTENDED MONITOR, DEBUGGER, DISASSEMBLER

SYSTEMS

QDP-8100 WITH 2 MEGABYTES STORAGE STANDARD (OPTIONAL 4 MEGABYTES)

- Z-8000 SERIES 16 BIT CPU S-100 BOARD - SEE ABOVE
- SOFTWARE (PROVIDED WITH SYSTEM)
  - CP/M 2.2[1] OPERATING SYSTEM
  - BASIC
  - Z80/8080 EMULATOR
  - MONITOR, DEBUGGER, DISASSEMBLER
  - SOFTWARE OPTIONS: PASCAL
  - UNIX[2] OPERATING SYSTEM COMING

$6,395.

QDP-100 WITH 2 MEGABYTES STORAGE STANDARD (OPTIONAL 4 MEGABYTES)

- Z-80 SERIES 8 BIT CPU S-100 BOARD (4 MHZ Z-80, DBL DENSITY DISK CONTROLLER, 2716 PROM BURNER 2 PARALLEL & 2 SERIAL PORTS, REALTIME CLOCK)
- SOFTWARE (PROVIDED WITH SYSTEM)
  - CP/M 2.2[1] OPERATING SYSTEM
  - BASIC
  - ACCOUNTS RECEIVABLE, GENERAL LEDGER, ACCOUNTS PAYABLE, PAYROLL WITH COST ACCOUNTING
  - OPTIONAL SOFTWARE: FORTRAN, PASCAL, COBOL, C

$4,995.

EACH SYSTEM CONTAINS:

- INTELLIGENT CRT TERMINAL (80 CHARACTERS X 24 LINES)
- 64 KBYTES RAM
- TWO 8 INCH, DOUBLE SIDED, DOUBLE DENSITY FLOPPY DISK DRIVES WITH CONTROLLER
- 2 SERIAL AND 1 PARALLEL (2 PARALLEL FOR QDP-100) PORTS
- ATTRACTIVE WOODGRAIN CABINET WITH POWER SUPPLIES AND CABLING

[1]CP/M™ DIGITAL RESEARCH
[2]UNIX™ BELL LABS

FULL TECHNICAL SUPPORT FROM THE STAFF AT QUASAR DATA PRODUCTS

4 Mhz 64K Dynamic RAM

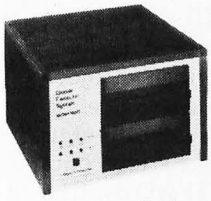
16K - $250.00 32K - $350.00 48K - $450.00 64K - $549.00

TELETEK DBL. DENSITY, DBL. SIDED

Disk Controller Board..... $395.00

QUASAR FLOPPY SYSTEM

- Two MFE DBL sided drives • Cable • Case & Power Supply assembled and tested Wood cabinet $1895.00

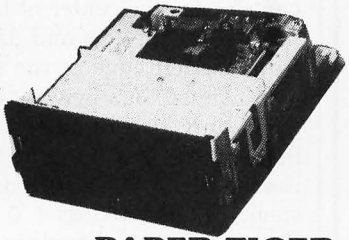


QUASAR 2 MEG FLOPPY

- 2 MFE double sided drives
- Teletek disk controller board
- Power supply & cable
- Wood cabinet
- CP/M version 2.2 & bios
- Assembled & tested $2295.00

Dealer Inquiries Invited, Hours: 9-5:30 M-F

Specifications Subject To Change



MFE Double Sided - Double Density 8" Floppy Disk Drives. (the best) $650.00
Using the Teletek Controller under CP/M, THIS DRIVE WILL GIVE YOU ALMOST ONE MEGABYTE PER DISK DRIVE.
Power supply for above $110.00

PAPER TIGER

Includes Graphics $949.00
Cable for TRS-80 ..... $39.00
Call for Apple

TI - 820

Serial Printer -
Full package options... $1995.00

UnixTM - Bell Lab 30 Day ARO CP/MTM - Digital Research

Checks, money orders accepted
Add $2.50 freight charges on orders under 10 lbs. Over 10 lbs. F.O.B. Cleveland

QUASAR DATA PRODUCTS

25151 Mitchell Dr., No.Olmsted, Ohio 44070 (216)779-9387

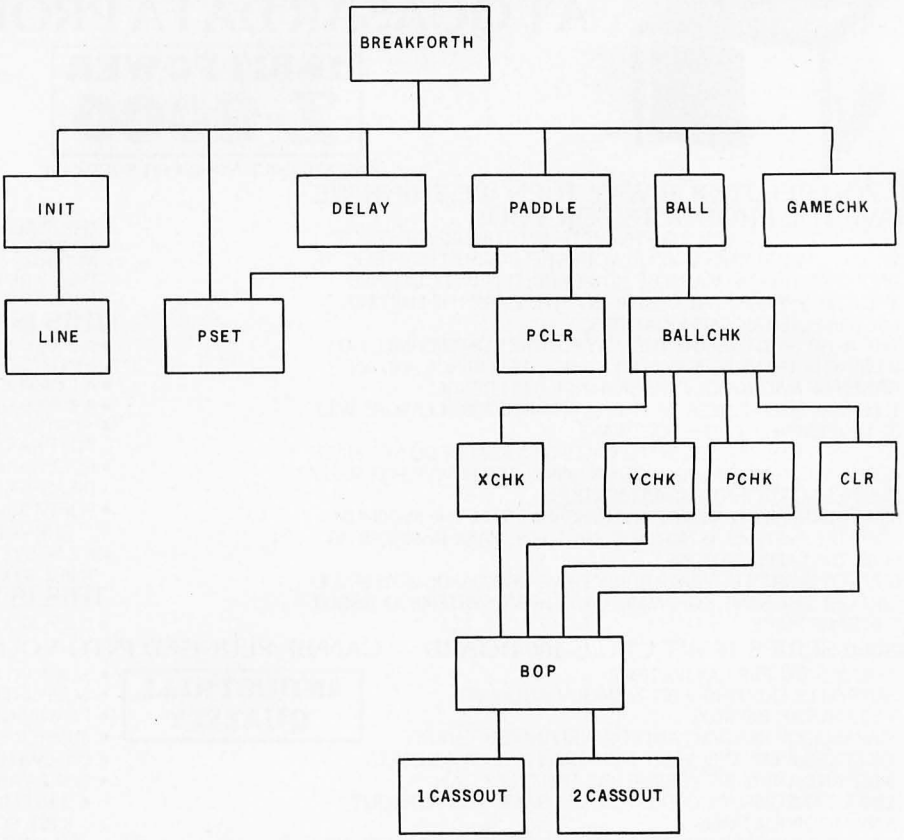


Figure 2: *A hierarchical diagram of the BREAKFORTH program. Each box contains a word used within the BREAKFORTH program and is used by the word(s) in the box(es) above it. See table 1 for a definition of each word.*

tween the word being defined and the first word of the definition; this adds to the clarity of the definition. PTC (for "put cursor") places the cursor at a given point on the screen, much like the PRINT@ instruction in TRS-80 BASIC. It expects two numbers on the stack, the row (second-to-top) and the column (on top) giving the desired position for the cursor. (For example, { 8 32 PTC } puts the cursor near the center of the screen, 8 rows from the top and 32 characters from the left edge of the screen.)

However, our new word LINE expects only one number on the stack because the first thing it does when it is called is to put a zero on top of the stack. So the words { 0 PTC } put the cursor at the beginning of a given line (that is, at position (x,0), where x is the number on top of the stack when LINE is called).

The FORTH word ECHO (EMIT in some other versions of FORTH) is like the PRINT CHR$ function in BASIC—it outputs the corresponding ASCII character for the number. In this case, { 30 ECHO } outputs a clear-to-the-end-of-the-line signal on the TRS-80. (By the way, the 30 is the decimal number thirty; although you can change to hexadecimal with the word HEX or to any other numeric base, MMSFORTH assumes decimal numbers unless told otherwise.)

Now we are finally able to say what the word LINE does: the phrase { x LINE } clears line x and leaves the cursor at row x, column 0. { 0 PTC } puts the cursor at the beginning of the line, and { 30 ECHO } clears the line with a special character (ASCII decimal 30) and leaves the cursor where it is.

The final word described in block 50, INIT , begins in line 8. Its definition is longer than most words, but its function is not at all mysterious once you know a few FORTH words. CLS clears the video screen (as in TRS-80 BASIC), { 0 LINE } clears line zero, and { " } ({ ". } in some FORTHs) causes the character string until the next quote mark to be printed, just as PRINT " *STRING* " does in BASIC. The word #IN causes a single-

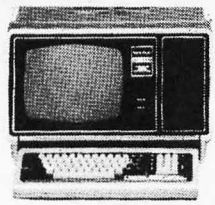
Text continued on page 158

ΩMEGA SALES CO.

"WHOLESALE COMPUTER PRICES" DIRECT TO THE PUBLIC

12 Meeting St., Cumberland, R.I. 02864

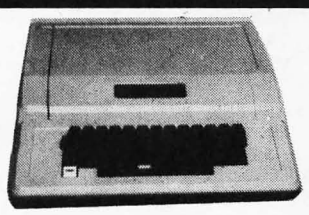
Circle 104 on inquiry card.



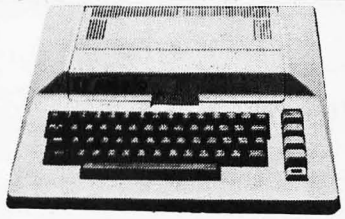
TRS•80
Model II · $3,500

PRODUCT SPECIAL OF THE MONTH!!

Products are NOW IN STOCK AT ΩMEGA Sales Co.



Apple II
16K · $1049



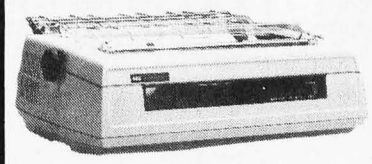
Atari 800 · $749

INTERTEC SUPERBRAIN
32K RAM - $2449.00
64K RAM - $2649.00

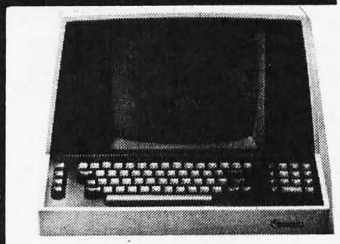
(Prices valid July 15 · Sept. 1, 1980)

For: TRS•80, Apple, CBM,
(Interface Included)

Epson TX-80 · $745



NEC Spinwriter
5510-5530 · $2449



Soroc 120 · $699

CALL TOLL FREE FOR ΩMEGA'S PRICE!

ΩMEGA OFFERS THE BEST DELIVERY AND PRICE ON:
APPLE • ATARI • TRS·80 MODEL II • INTERTEC •
T.I. 810 • HEWLETT-PACKARD-85 • SOROC •
COMMODORE • NEC • QUME • CENTRONICS

ΩMEGA sells only factory fresh, top quality merchandise to our customers.
ΩMEGA will try to match any current advertised price with similar purchase conditions.
Before you buy anywhere else - be sure to call ΩMEGA Sales Co.
1-401-722-1027 or

ΩMEGA TOLL FREE 1-800-556-7587

ΩMEGA ships via UPS, truck, or air. COD's, VISA, Mastercharge accepted.
"A member in good standing of the better business bureau."

VISA master charge

Listing 1: *The BREAKFORTH program. These six blocks, when loaded into an MMSFORTH system, cause the BREAKFORTH program to compile, execute, and, once finished, erase itself from the system. Tape-based users should omit the last three words in the last block. This program does require that the MMSFORTH words for random numbers (block 32 on the MMSFORTH system disk or cassette) and for TRS-80 graphics (block 33) be available to the FORTH system. If these blocks have already been loaded, delete the two LOAD commands in block 50, line 3. Also, the sequence { A MVI 255 } in lines 10 and 11 of block 51 is the notation FORTH uses for the 8080 assembly-language statement MVI A,255.* [To speed up paddle response, you can replace the 3 in block 55, line 8 with a higher value. Personally, I enjoy playing the game at speed level 1, with a 12 replacing the 3....**GW**]

```
BLOCK : 50

 0 ( BREAKFORTH/MMSFORTH, BY ARNOLD SCHAEFFER, PART 1 OF 6          )
 1 (     COPYRIGHT 1980 BY MILLER MICROCOMPUTER SERVICES            )
 2 (  W/SOUND - USE THE LEFT AND RIGHT ARROWS TO MOVE THE PADDLE   )
 3   : TASK ; 32 LOAD ( RANDOM #'S ) 33 LOAD ( GRAPHICS ) RANDOMIZE
 4   0 CVARIABLE SPEED  0 CVARIABLE SPVAR  0  VARIABLE SCORE
 5   0  VARIABLE XPOS   0  VARIABLE YPOS   2  VARIABLE PPOS
 6   1  VARIABLE YDIR   1  VARIABLE XDIR   0  VARIABLE BEST
 7 : LINE  0 PTC 30 ECHO ;
 8 : INIT  CLS 0 LINE " SPEED ( 1 - 10, 1 IS FASTEST )"
 9         #IN 1 MAX 10 MIN 10 U* SPEED C!
10         0 LINE " NUMBER OF BALLS DESIRED"   #IN
11         CLS  64 0 DO 3 I DSET 4 I DSET LOOP
12         48 3 DO I 0 DSET I 63 DSET I 1 DSET I 62 DSET LOOP
13         191 15616 320 FILL   0 SCORE !
14         0 LINE " BREAKFORTH IN MMSFORTH    SCORE: 0     BEST:"
15         BEST ?    0 54 PTC " BALL:"  ;



BLOCK : 51

 0 ( BREAKFORTH/MMSFORTH, BY ARNOLD SCHAEFFER, PART 2 OF 6 )
 1
 2 : PCLR   32 PPOS @ 16320 + 8 FILL ;
 3 : PSET  176 PPOS @ 16320 + 8 FILL ;
 4
 5 : PADDLE
 6   14400 C@ 32 = IF PCLR -1 PPOS @ +  2 MAX PPOS ! PSET THEN
 7   14400 C@ 64 = IF PCLR  1 PPOS @ + 54 MIN PPOS ! PSET THEN
 8 ;
 9
10 CODE 1CASSOUT   1 A MVI 255 OUT NEXT    ( THESE 3 LINES      )
11 CODE 2CASSOUT   2 A MVI 255 OUT NEXT    ( PRODUCE THE SOUND. )
12 : BOP   10 0 DO  1CASSOUT  2CASSOUT  LOOP ;
13
14
15



BLOCK : 52

 0 ( BREAKFORTH/MMSFORTH, BY ARNOLD SCHAEFFER, PART 3 OF 6 )
 1
 2 : XCHK
 3   XPOS @  2 < IF XDIR @ MINUS XDIR !  2 XPOS !  BOP THEN
 4   XPOS @ 61 > IF XDIR @ MINUS XDIR ! 61 XPOS !  BOP THEN
 5 ;
 6
 7 : YCHK
 8   YPOS @  5 < IF 1 YDIR !  5 YPOS !  1 SPVAR C!  BOP  THEN
 9   YPOS @ 23 < IF SPVAR C@ 4 MIN SPVAR C!  THEN
10   YPOS @ 19 < IF SPVAR C@ 3 MIN SPVAR C!  THEN
11   YPOS @ 15 < IF SPVAR C@ 2 MIN SPVAR C!  THEN
12 ;
13
14
15



BLOCK : 53

 0 ( BREAKFORTH/MMSFORTH, BY ARNOLD SCHAEFFER, PART 4 OF 6 )
 1
```

Listing 1 continued on page 158

Text continued from page 154:

precision number to be entered from the keyboard and placed on top of the stack. The phrase { 1 MAX } causes the number to be replaced by 1 if the number just entered is smaller. Similarly, the phrase { 10 MIN } limits the number on the top of the stack to a maximum value of 10. { 10 U* } multiplies the number by 10 (U* is an unsigned single-precision multiply), and { SPEED C! } stores the value from the top of the stack in the single-byte variable SPEED .

Each of the above phrases contains a number and an operation. Since each operation requires two numbers on the stack, the number entered by #IN is the first number, with the second number always being supplied by the first word of the phrase.

Using the same words as listed above, line 10 again clears line 0, prompts for the number of balls to be used in the game, putting that number on top of the stack with the word #IN .

Line 11 clears the video screen again and sets up the back (top) wall of the BREAKFORTH "court" using a do-loop and double-width graphics. In FORTH, the parameters of the loop go on the stack before the loop is called, so { 64 0 DO } begins the loop, and the word LOOP ends it. The loop will be executed sixty-four times, and the word I puts on top-of-stack the current value of the loop (0, 1, 2, 3, ... ,63); note that I does not take on the limit value of 64. The phrase { 3 I DSET } sets a double-width character at row 3, (double-width) column I; similarly, { 4 I DSET } sets the double-width character on the next row below the first.

Similarly, line 11 sets the right and left walls of the BREAKFORTH court, columns 0 and 1 for the left wall and columns 63 and 64 for the right wall.

The phrase { 191 15616 320 FILL } in line 13 creates the initial wall of bricks by using character code decimal 191 (a whited-out character cell) to fill an area of *memory* (the video display area of the TRS-80) starting at location 15616 and filling for a total of 320 bytes.

The phrase { 0 SCORE ! } , also in line 13, shows us how we store a value (0) in a variable (SCORE) by using the *store* operator { ! } . Two points should be mentioned here. First, executing a variable name (like SCORE) causes the *address* of the variable, *not* its value, to be pushed onto the top of the stack. Second, the *store* operator { ! } requires the value to be the second-to-top item in the stack and the address of the variable receiving the new value to be the top item in the stack.

The words in line 14 clear line 0 and print a message on the same line, setting the score to zero but leaving the cursor just after the colon that ends the message.

In line 15, the phrase { BEST ? } causes the value of BEST to be displayed on the screen, and the rest of line 15 completes the message that is shown on line 0 of the screen. Finally, the semicolon on line 15 ends the definition of INIT begun on line 8.

Listing 1 continued:

```
 2 2 CONSTANT 2   -2 CONSTANT -2
 3
 4 : PCHK  0  YPOS @ 47 >=
 5   IF 46 YPOS !  XPOS @ PPOS @ - DUP 0 >= OVER 8 < AND
 6      IF  -1 YDIR !   BOP
 7         NCASE 0 1 2 3 4 5 6 7 " -2 -1 -1 -1 1 1 1 2 CASEND
 8         XDIR !
 9      ELSE DROP 1+
10      THEN
11   THEN
12 ;
13
14
15
```

```
BLOCK : 54

 0 ( BREAKFORTH/MMSFORTH, BY ARNOLD SCHAEFFER, PART 5 OF 6 )
 1
 2 : CLR
 3   XPOS @ 2 - 124 AND 2+ DUP 4 + SWAP DO YPOS @ I DCLR LOOP
 4   YPOS @ 27 - ABS SCORE +!  0 32 PTC SCORE ?   BOP
 5   YDIR @ MINUS YDIR !
 6 ;
 7
 8 : BALLCHK  YDIR @ YPOS +!  XDIR @ XPOS +!  XCHK  YCHK  PCHK
 9            YPOS @ XPOS @ D? IF  CLR  THEN
10 ;
11
12 : BALL  YPOS @ XPOS @ DCLR
13         BALLCHK  DUP 0= IF  YPOS @ XPOS @ DSET  THEN  ;
14
15 : GAMECHK  SCORE @ 1800 MOD 0= IF  191 15616 320 FILL  THEN  ;
```

```
BLOCK : 55

 0 ( BREAKFORTH/MMSFORTH, BY ARNOLD SCHAEFFER, PART 6 OF 6 )
 1 : DELAY SPEED C@ SPVAR C@ U* 0 DO LOOP ;
 2 : BREAKFORTH
 3   BEGIN INIT 0    PSET
 4      DO  2000 SPEED C@ / 0 DO  DELAY  PADDLE  LOOP
 5         0 60 PTC  I 1+ .    5 SPVAR C!
 6         2 RND 1 = IF 1 ELSE -1 THEN  XDIR !   1 YDIR !
 7         58 RND 2+ XPOS !  29 YPOS !
 8         BEGIN  3 0 DO PADDLE LOOP
 9            BALL  GAMECHK  DELAY
10         END
11      LOOP    SCORE @  BEST @  MAX BEST !
12      8 18 PTC "  RUN GAME AGAIN "    Y/N
13   END
14 ;
15 BREAKFORTH   FORGET TASK DIR
```

The Middle Blocks

Whew, that was a lot of explaining! Now you see why FORTH is not very easy for beginners to read—you are packing a lot of work into a small space, using an ever-more-specialized

But, do you know all the compo[illegible]on Tarbell has ready for you?

When someone says "Tarbell" there's no doubt what's meant . . . the cassette interface whose reliability and solid engineering made it an industry standard.

Since that first breakthrough-product, Don Tarbell has expanded his list of useful, dependable components . . . components to meet your needs of today, and keep you prepared for tomorrow.

Check this partial list of quality components Don Tarbell has ready for you. You're probably ready for them, right now.

- When it comes to RAM memory, Tarbell means reliability. 16K and 32K static memory that offers you easier trouble shooting, and far *easier* maintenance. Remember that.

- Tarbell BASIC brings simplicity and sophistication to your programs. Our BASIC is easier to program, and offers unique commands and statements not found in regular BASICS under any name.

- CP/M® disk operating system is, of course, the standard for software exchange. At Tarbell we provide our own approved CP/M system modified for all Tarbell floppy disk interfaces. Note. We also have MP/M® for those interested in multi user systems.

- The Tarbell VDS line comes as a complete package . . . or, as separate units. For example, the Tarbell mainframe can be ordered with 1 or 2 Shugart or Siemens drives, or no drives. Whichever way you go, you get the reliability of Tarbell tested components.

- With the Tarbell Double Density floppy disk interface, storage capacity, speed and versatility are greatly increased. Under our DD CP/M, single and double density disks may be intermixed with no penalty. The system automatically determines which is in place.

We also still have our Single Density floppy disk interface. It's specifically designed to operate with many different and unusual drives. Naturally, they're Tarbell tested.

Tarbell Electronics
950 Dovlen Place, Suite B
Carson, California 90746
(213) 538-4251 / 538-2254

*CP/M & MP/M are products of Digital Research Corp.

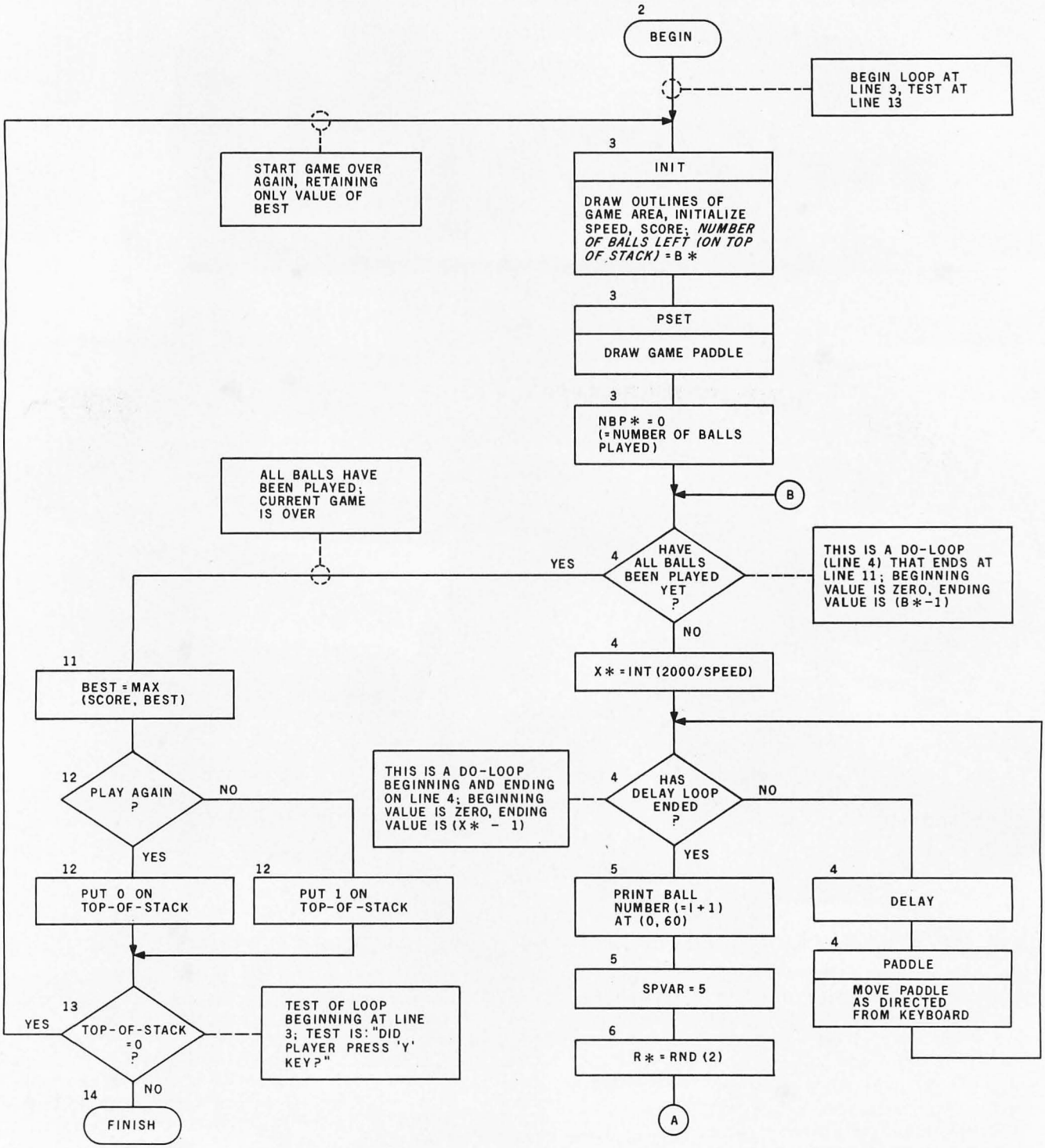
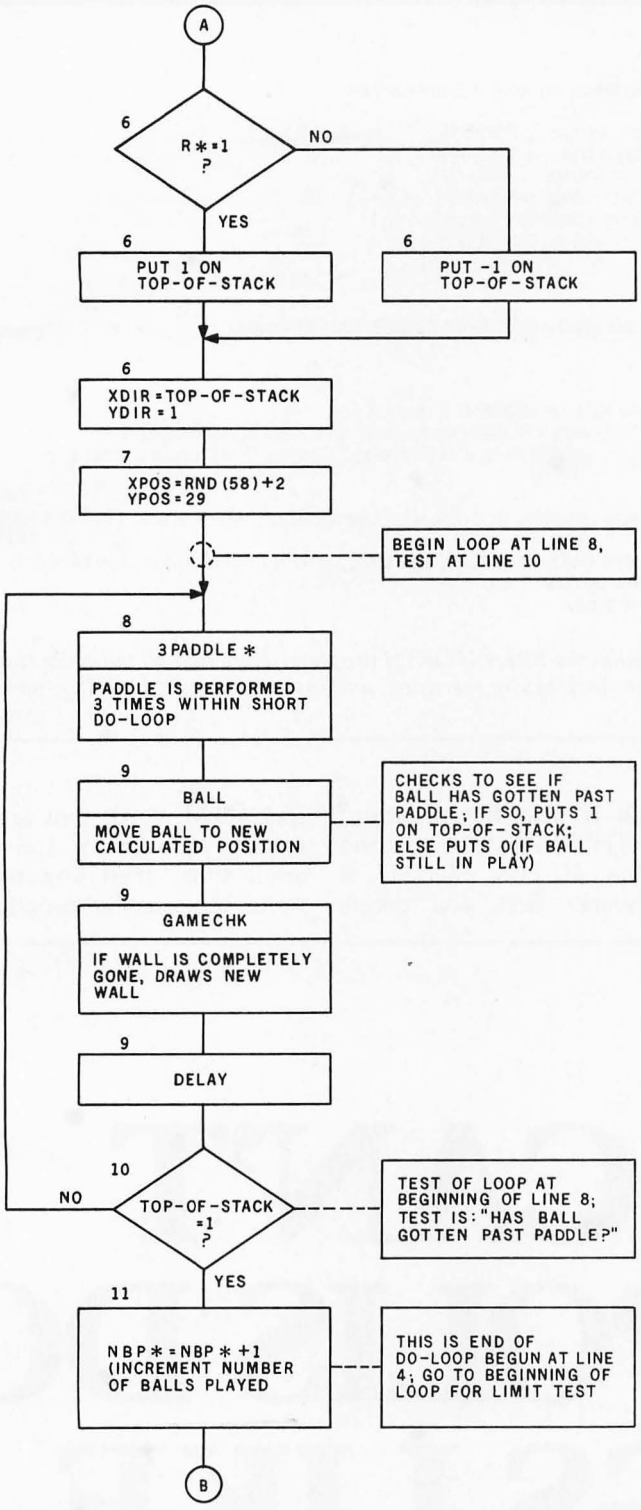


Figure 3: *A flowchart for the BREAKFORTH program (given in listing 1, block 55). The number above each box is the line number within block 55 that performs the action of the box. Many calculations in FORTH are done on the stack and do not acquire variable names. Because of this, an asterisk in a variable or procedure name (eg: X*, 3PADDLE*) denotes that the name was given only in this flowchart to add clarity.*

instruction set. Experience with reading and writing FORTH code makes the process easier, but spacing, indentation, use of descriptive word names, and lots of comments are always helpful. A surprise to the BASIC user: none of these source-code editing improvements use any extra programmable memory space.

Table 1 explains much of what the words in blocks 51 thru 54 do, but let us look at some of the interesting features contained in these lines of FORTH code.

When the ten-to-twenty times speed increase of FORTH over BASIC is not enough (or when we want to do things that cannot be done with existing FORTH words), we can redefine some FORTH words in the assembly language of the computer



(in the case of the TRS-80, 8080 or Z80 assembly language). When we want a FORTH word (program) to run faster, usually a short assembly-language definition of the word that gets used the most will speed things up sufficiently. Lines 10 and 11 of block 51 are the only two words used in BREAKFORTH that are defined in 8080 assembly language. (MMSFORTH comes with a compact 8080 assembler built in, like many Z80-based FORTHs. A *full* Z80 assembler also is available from MMS at a modest price.)

Inspection of lines 10 and 11 of block 51 shows that assembly-language definitions begin with the word CODE (instead of { : }) and end with the word NEXT (instead of { ; }). Here, FORTH's 8080 assembler is used to define a new type of word to output to a port. Both 1CASSOUT and 2CASSOUT drive the cassette recorder port (I/O port 255 on the TRS-80), and the word BOP executes both these words in a do-loop ten times to create a short square-wave sound on the external speaker.

The definition of PCHK ("paddle check" of ball location) in block 53 uses two more constructs. There are two *if* constructs, the inner one beginning in line 6 and ending in line 10, the outer beginning in line 5 and ending in line 11. (Notice that only the inner loop uses the optional *else* clause, as in line 9.) The second construct is a numeric *case* construct, NCASE ; as shown in line 7. When NCASE is executed, it expects the number on top of the stack to be one of the numbers listed between NCASE and the double quote marks (here, zero thru seven). The value found causes the execution of the corresponding FORTH action word in the series of apparent numbers between the double quote mark and the word CASEND. (Numbers are words but are not in FORTH's dictionary—when they are "executed," they are pushed on top of the stack. MMSFORTH case statements require their action words to be words in the FORTH dictionary and not numeric literals, so in block 53, line 2, 2 and −2 are defined as constants (FORTH words). 1 and −1 are already defined as constants by standard FORTH. Taking { 2 CONSTANT 2} as an example, the first 2 is the value of the constant, while the second 2 is the name of the constant; we might have used the word TWO in its place.) In our program, { 0 NCASE } causes the word −2 to be executed. { 1 NCASE } , { 2 NCASE } , or { 3 NCASE } cause −1 to be pushed on top of the stack, and so on. Only one of the words is executed; execution then continues with the first word after CASEND . MMSFORTH also has an alpha-numeric case statement that branches on the value of a single character. Each may be thought of as a compact, structured, many-branched alternative to a nested series of *if* statements.

The Last Block

Block 55, the last block used to

Word Name	Usage
SPEED	CVARIABLE contains speed of play.
SPVAR	CVARIABLE contains speed multiplier, depends on height ball reaches.
SCORE	VARIABLE contains current score.
XPOS	VARIABLE contains current ball X position (range, 2 thru 61).
YPOS	VARIABLE contains current ball Y position (range, 5 thru 47).
PPOS	VARIABLE contains current paddle position (range, 2 thru 54).
XDIR	VARIABLE contains current ball X increment (possible values: −2, −1,1,2).
YDIR	VARIABLE contains current ball Y increment (possible values: −1,1).
LINE	Expects *n* on top of stack; moves cursor to line *n*, clears line.
INIT	Asks questions and draws display.
PCLR	Clears paddle.
PSET	Draws paddle.
PADDLE	Checks for right- or left-arrow key being pressed and moves paddle appropriately.
1CASSOUT	8080-code procedure for sound.
2CASSOUT	8080-code procedure for sound.
BOP	Makes one bounce noise.
XCHK	Checks if ball hit either side wall, modifies XDIR and XPOS if necessary.
YCHK	Checks if ball hit top wall and modifies YDIR and YPOS if necessary; also sets speed multiplier.
PCHK	Checks if ball at paddle level; if so, did it hit paddle or is it out of play? Leaves F on top of stack; F = 0 if ball still in play, else 1.
CLR	Clears brick, modifies score and YDIR.
BALLCHK	Increments ball position and checks for wall, paddle, or brick hits. Leaves F on top of stack; F = 0 if ball still in play, else 1.
BALL	Clears old ball position, calls BALLCHK, and draws new ball; see BALLCHK for value left on top of stack.
GAMECHK	Checks if all bricks cleared and draws new barrier if so.
DELAY	Causes a given time delay between ball moves.
BREAKFORTH	Main game loop.

Table 1: *Table of variable names and FORTH words used in the BREAKFORTH program. Note that all variables leave their address on the stack, that LINE removes one entry from the stack before executing, and that PCHK , BALLCHK , and BALL add one entry to the stack after executing.*

define the word BREAKFORTH , defines one last word (DELAY , in line 1), then puts all the words defined so far together to define the word (which is also the program) BREAKFORTH . This is a good demonstration of how FORTH is meant to work: first you define specialized words that are helpful in solving problems of a given class or application, then you use them to write the specific program needed.

WHY CAN'T MICROPOLIS DO THINGS LIKE EVERYONE ELSE?

(The building words, if chosen and defined properly, can be used to help write other programs in the same class.)

The word BREAKFORTH is defined in lines 2 thru 14. A flowchart for the program is given in figure 3; the number to the left of each box gives the line number within block 55 which the box is associated with.

Line 15, the last line of block 55, is interesting in that it triggers all the work done so far. The word BREAKFORTH causes the definition of the word to be executed. Once the game is finished, the next words, { FORGET TASK } , are executed; these words cause the word TASK (remember block 50?) and every word defined after it to be erased from the vocabulary of the language. This is done to free up the computer once we are finished playing BREAKFORTH. You can omit these words if you wish, but the disk program is recalled into memory so easily (with the phrase { 50 6 LOADS }) that most people prefer to keep the FORTH dictionary as uncluttered as possible. The last word, DIR , causes the standard disk MMSFORTH directory to be displayed on the screen. (The last three words should be deleted if you are running the cassette version of MMSFORTH.)

Summary

It takes some work to understand your first FORTH program. But this work is only the flip side of the same coin that makes FORTH such a powerful language—where else can you easily write such a large and speedy program in such a small space? [*The only other candidate language I can think of is APL, which is also known for its compactness and unreadability to the uninitiated. . .* GW] But, of course, your second FORTH program is easier than your first, and so on. Better yet, your second program may be 90% written by your first, thanks to FORTH's structured and modular design.

We hope you have enjoyed this introduction to FORTH. We can assure you that it has just scratched the surface of FORTH, which performs equally well in process control projects and business applications. FORTH improves our programming skills while improving our computer's effective speed, memory capacity, and instruction set. It is a most satisfying language. ■

Miller Microcomputer Services offers a number of products and services based on the FORTH language. Version 1.9 of MMSFORTH, the language used in this article, runs on a 16 K-byte or larger TRS-80 Model I with Level II BASIC. The disk version is $79.95, and the cassette version is $59.95. Each package contains the complete MMSFORTH system (including a full-screen editor and an 8080 assembler), FORTH source code, documentation, and the microFORTH PRIMER *book from FORTH Inc.*

For further information, send a self-addressed, stamped business envelope to:

MMSFORTH Information
Miller Microcomputer Services
61 Lake Shore Rd
Natick MA 01760

FORTH Extensibility

Or How to Write a Compiler in 25 Words or Less

Kim Harris
1055 Oregon Ave
Palo Alto CA 94303

A computer language should help users solve problems. Languages bridge the gap between the primitive operations the computer can perform (add, fetch from memory, etc), and the tasks a user needs (invert a matrix, search a file, etc). When the operations of an application are well matched to those of a language, the solution can be simplified and developed in less time; in addition, the resulting program becomes more readable.

Because all applications have various needs, it is impossible for a nonextensible computer language to satisfy all needs equally well. Although languages have been produced which attempt to include all possible operations, structures, and facilities, these have not been satisfactory.

FORTH's approach is to provide a few techniques that allow a user to quickly add the special operations his particular application requires. The remainder of this article will describe some of these techniques and give examples that add arrays (with and without subscript range checking), virtual arrays, and a case selection control structure.

Extending the Language

The ability to add language facilities and compiler structures is called *extensibility*. FORTH is extensible on three levels of increasing power:

- using existing compilers
- creating new compilers
- creating new operating systems

Editor's Note

*In this article, Kim Harris uses the syntax of FORTH-79, which is different from that of existing FORTH implementations, for his examples. FORTH-79 is a standard set of FORTH words that, if used to build all other FORTH words needed for a given application, insures the complete portability of a given program between different versions of FORTH. Members from FORTH Inc, the FORTH Interest Group, the European FORTH Users' Group, and MMS worked together to define FORTH-79. I have noted the differences between the text and existing FORTH implementations (in particular, fig-FORTH and MMSFORTH) where known....***GW**

This article focuses on the second level and demonstrates the construction and use of specialized compilers. The specialized compilers are usually simple (definable in a few source lines), but permit entire new classes of language or compiler facilities to be added to a FORTH system.

The compilation of any computer language is diagrammed in figure 1. Compilation is the process of converting a source language program into a form that a computer can use.

FORTH uses multiple compilers to implement different compiler functions. For example, compiling a data structure declaration (eg: an array) is distinctly different from compiling an executable statement. FORTH uses separate compilers for these two activities. Such compilers are many times simpler than the compilers for most popular languages (eg: BASIC, Pascal, COBOL); however, a collection of FORTH compilers can perform all the functions of the other languages' compilers (when these functions are adapted to a FORTH-like environment).

FORTH uses the English word "word" to mean an executable procedure, not a piece of memory. In this article, "word" will be used in the FORTH sense, and storage sizes will be specified in terms of 8-bit bytes.

User-Defined Words

The input language to the FORTH compilers is a sequence of FORTH source language word-names separated by spaces. (Unlike other languages, a space in FORTH is *very* important.) The output is one *dictionary definition* for each new word (procedure) compiled. The compilation process is controlled by special FORTH procedures called *defining words*. A *source definition*, which is a series of FORTH words including defining words, specifies a procedure that can be compiled by executing (typing in) the sequence. The result of compilation is a

Figure 1: *Compilation of any computer language. A program in some computer language is input to a compiler. The compiler produces a functionally equivalent program in a different, object language.*

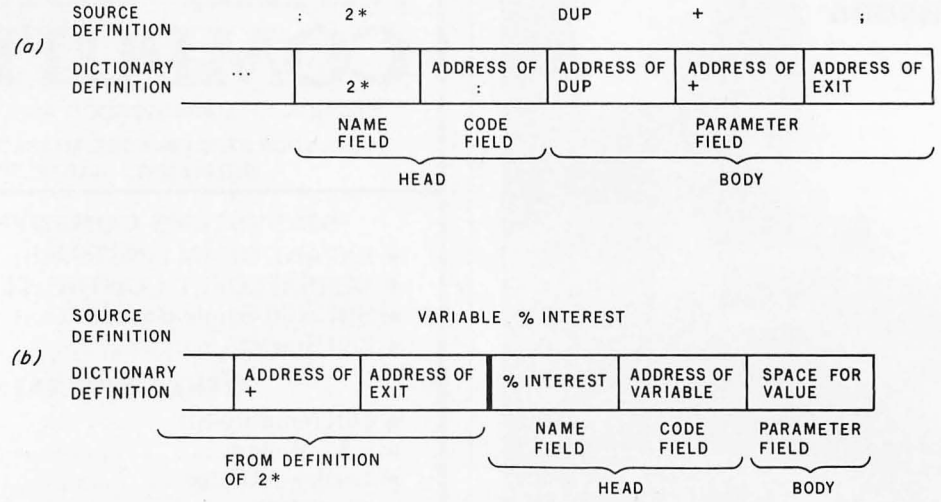


Figure 2: *Examples of extending the FORTH language. The first source line adds a new operator named 2* (see figure 2a); the second source line adds a new operand named %INTEREST (see figure 2b).*

dictionary definition, which is a block of FORTH-interpretable instructions. All compiled FORTH words are kept in this dictionary, which is usually located in the computer's memory.

User-defined words are treated the same as system-supplied words. If some new words are defined which behave like operators (eg: triple-precision versions of the FORTH words + , − , * , / , etc), then the language has been truly extended to include these operators. Subsequent words may use these new words as system-supplied operators.

Examples of standard, system-supplied defining words are { : } (colon), which starts the compilation of subroutine-like procedures, and VARIABLE , which compiles a named memory location for the variable's value.

A source definition consists of a defining word followed by the name of the word being defined and then by other FORTH words and numbers. Figure 2 illustrates the source definitions and the corresponding dictionary definitions for two new words named 2* and %INTEREST . (FORTH word-names may be made of *any* nonblank characters.) The word 2* simulates a multiplication by 2 by adding a value to itself.

The defining word { : } compiles the words that follow it in a definition, which is then added to the dictionary. Each FORTH dictionary definition consists of two parts: a *head* and a *body*. The head contains system-internal information including a *name field* and a *code field*. (A *link*, which points from a definition to a previous definition, is part of the head but will be ignored in this article.) The name field contains the name of the word. The code field contains a pointer to the instructions that will be executed when the word is executed.

For definitions compiled by { : }, the code field points to a procedure that begins the execution of the words referenced in the definition. The body of this kind of definition, called the *parameter field*, is a series of addresses that point in order to each FORTH word in the definition. The addresses of these referenced words are placed in the parameter field by the { : } compiler, and the definition is ended by the FORTH word { ; } (semicolon). The execution of the word EXIT (compiled at the reference to { ; }) ends the execution of the word.

Some Examples

The word 2* will leave a result that is twice the value of its input. (See figure 2a.) Examples in this article will underline the input typed by the user and will end in an unseen carriage return; the computer's response follows. The following line shows the use of the word 2* :

```
3 2* .  6 OK
```

The use of 2* causes the words in its definition to be executed, as if the user had typed:

3 DUP	two copies of 3 on the stack
+	add both 3s
.	print result from top of stack

Any subsequently compiled word may call the word 2* as if it were any other FORTH word. When called, 2* performs its function and then returns. This is analogous to the execution of a subroutine call in other languages.

A word is called by simply using its name, as in the following source definition for 4* .

```
: 4*  2* 2* ;
```

The defining word { : } has been used to compile another definition into the dictionary.

Using 4* will cause 2* to be called and executed twice. Here is an example of the use of the word 4* .

```
3 4* .  12 OK
```

The second word defined in figure 2 uses the defining word VARIABLE to compile a dictionary definition that contains data. The source word-name %INTEREST is compiled into a new dictionary definition containing a

Level	Method
I	Using standard FORTH defining words to add new operations (programs).
II	Creating new user-defined defining words that, in turn, create new classes of words.
III	Creating new FORTH-like systems through metaFORTH.

Table 1: *Levels of extensibility in FORTH. Level I refers to the act of defining ordinary words in FORTH using standard defining words. Level II refers to the creation of new defining words that are then used to create a family of ordinary FORTH words. Level III refers to the act of altering and recompiling FORTH itself (sometimes called metaFORTH) to create significantly different variant FORTH-like systems. Higher levels imply greater capability and flexibility.*

2-byte area where the value of the variable will always be stored. (The use of the word-name %INTEREST , either inside or outside a definition, will cause the *address* of this variable's value to be returned, not the value of the variable.)

The dictionary definition for %INTEREST contains the variable's name, a pointer to the instructions executed when %INTEREST is executed, and a 2-byte data area. The code fields of all words defined by VARIABLE point to a procedure which returns the address of the data area of the variable when the variable's name is referenced. All FORTH words, even data words, have some code that is executable.

The two defining words of this figure are actually different compilers. The defining word { : } compiles procedure definitions, while the defining word VARIABLE compiles data definitions. All user-added operators and operands can be used exactly like the system-supplied ones. Even new control structures can be added to the FORTH compiler by the user.

Levels of Extensibility in FORTH

As shown in table 1, there are three levels of extensibility supported by FORTH. The two words defined in figure 2 are examples of extensibility level I, the most commonly used level. It comprises the "ordinary" act of programming in FORTH. Although it is very useful, this level is the most restrictive and the least powerful of the three.

The process of writing and using new defining words is the second level of extensibility. Level II, which is more powerful than level I, allows a new "family" of words to be added to the language or compiler. This is done by creating a special word, called a *defining word*, that will be used to create FORTH words in the same family. The user specifies via the defining word how the compilation of a new family member (itself an ordinary FORTH word) is to be performed and what the result will be. Also the user specifies what a member of the family will do when it is executed.

Level III, the highest level of extensibility, is called *metaFORTH*. It uses the entire FORTH system to compile a collection of source definitions (including both lower levels) in order to produce a clone or a mutation of FORTH.

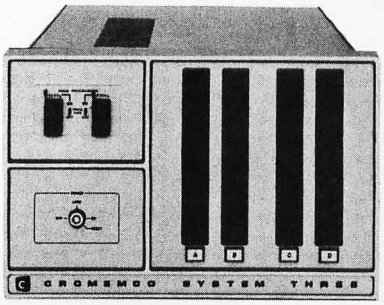
Circle 111 on inquiry card.

Circle 112 on inquiry card.

DISCOUNT PRICES

Microcomputers & Peripherals

**BITS
BYTES
BOOKS
BARGAINS**



**Cromemco • SWTPC • Lear-Siegler
Hazeltine • RCA • North Star
Verbatim • Perkin Elmer and others**

Fast, off the shelf delivery.
Call TOLL FREE 800/523-5355

MARKETLINE SYSTEMS, Inc.
2337 Philmont Ave., Huntingdon Valley, Pa. 19006
215/947-6670 • 800/523-5355
Dealer Inquiries Invited

APPLE II® DISK SOFTWARE
DATA BASE MANAGER -
IFO PROGRAM

The IFO (Information File Organizer) can be used for many applications such as sales activity, inventory, check registers, balance sheets, client/patient records, billing and much more. This can be accomplished easily and quickly without prior programming knowledge.

Up to 1000 records with a maximum of 20 headers and 10 report formats can be stored on a diskette. Information can be sorted and searched (3 levels). Mathematical functions can be performed to manipulate the information. Subtotals and totals can be calculated on any numeric field.

Requires 48K and Applesoft II on ROM (or Apple II Plus). Accommodates serial/parallel printers. Error protection devices provided. Program diskette and instruction manual - $100.

MAILING LIST PROGRAM - Print labels sorted or searched by 6 fields. On-screen editing. Line up routine. $40.00

Inventory Program – $140
Payroll Package – $240 (Specify state)
Apartment Manager – $325
Professional Time & Billing – $325
Speed Reading – $100

Send check/money order to:
SOFTWARE TECHNOLOGY for COMPUTERS (STC)
P.O. Box 428
Belmont MA 02178

or available from your local dealer

CONTRACT PROGRAMMERS
$15 to $30 per Hour

Our clients have immediate short-and long-term assignments available for experienced programmers in either field -- mini/mainframe. Paid weekly; full benefits available.

- Software Tech. Writers
- Software/Hardware Engineers (INTEL 8085)
- Programmer/Analyst (COBOL, IBM, or DEC 10)
- Systems Programmer (Mini/ Micro Assembly, FORTRAN, & BASIC plus)

**digital arts group
CONTRACT
SERVICES**

For immediate consideration, contact: Jim Barry, Suite 101.

Nine Bedford Street
Burlington, MA 01803
(617) 273-2780

MicroByte Software

2415-C Gateway Plaza
Crabtree Blvd.
Raleigh, North Carolina 27604

AT LAST! A fully implemented computer based file management system. . . Only a few minutes of instruction and you are creating and using your own client lists, mailing lists, inventories, bibliographies, vendor lists, and more.

DBMS80

Files, lists, or records, with user defined formats, can be created, sorted, edited, and printed with ease. Sub-files can be created out of parts of existing files, selecting parts of a record or individual records by a search criterion.

ALSO available with DBMS80. . .

REPORT80

Build your own custom defined and formatted reports and data summaries. Print lables with user specified formats that will fit your own forms.

DBMS80 and REPORT80 will run under either CP/M or TRSDOS

DBMS80 ..... $250.00
REPORT80 ..... $100.00
Manuals each $25.00

OTHER PRODUCTS OF MICROBYTE SOFTWARE:

EDIT80 ..... **$100.00**
Text editor and print formatter which runs under CP/M or TRSDOS
DISK80 ..... **$50.00**
Utility which allows you to examine and patch a disk.
UTILS **$50.00**
Apple PASCAL utilities: extensions to Apple Pascal, together with file control utilities, cross-reference, etc.
PAYROLL **$100.00**
Apple PASCAL payroll for 150 employees, full deduction options, etc.

Write or call today for further details on our products.
Source ID#**TCE373** (919) 833-4894

APPLE is a trademark of Apple Computer Corp.
TRSDOS is a trademark of Tandy Corp.
CP/M is a trademark of Digital Research

Circle 113 on inquiry card.

Circle 114 on inquiry card.

(Please don't be misled by my use of the word "compiler." I have been asked, "Can you write a compiler in FORTH that will compile BASIC, Pascal, COBOL...?" The answer is not easy. Defining words *can* compile application-oriented languages, but those languages should be FORTH-like in nature. Ordinarily, the language being compiled satisfies the syntax of FORTH—words separated by spaces. The compilation will result in FORTH-interpretable instructions that will add to its dictionary of word definitions.

In keeping with the FORTH philosophy of keeping all definitions small, defining-word definitions are also small. This results in compilers (defining words) that are simple and specialized, although the range of complexity of these compilers can vary greatly. A simple defining word such as VARIABLE may accept only one source word and produce a single, simple definition in the dictionary. A more complex defining word such as { : } may take several source words and produces a more complex definition.)

The remainder of this article concentrates on level II, defining new families of words. The scope and usefulness of new defining words are discussed using functional descriptions and examples. New defining words can be created which can later compile application-oriented languages.

Creating Families of Words

The technique of creating new defining words permits

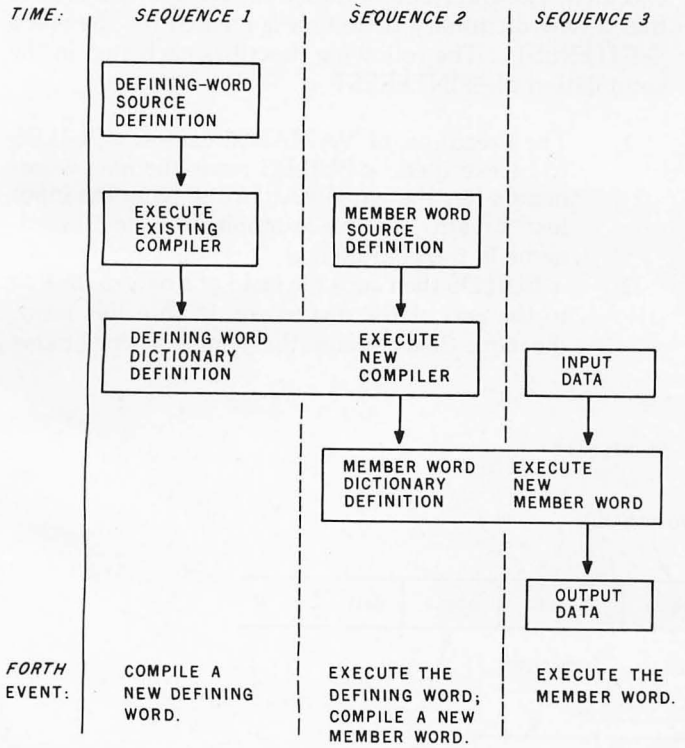


Figure 3: *The order of events governing defining words. The first event creates a word that will define a new family of words; this family currently has no members. The second event uses this new family-defining word to create a new family member, a named FORTH word. The third event occurs when any named FORTH word belonging to this family is used.*

a user to later create a family of FORTH words that can have any number of members. Each member shares some family traits but can also have individual characteristics. The family members are all the words that have been compiled by a defining word. Their common traits are specified by the defining word. However, each word in the family has individual characteristics that are assigned when added to the family.

For example, the defining word VARIABLE defines a family with individual members, each of which has a different name and value, but all share the same execution trait: specifically, the use of the name of any variable returns the address of its value.

It is important to understand that there are three time-ordered events related to defining words. These are listed in figure 3. These events will be explained using an example.

The compilation of the new words in figure 2 is a sequence 2 event (ie: using a defining word to compile another word). When the defining word VARIABLE is executed, as in:

VARIABLE %INTEREST

the source word %INTEREST is compiled.

Storing a value into the variable is a sequence 3 event.

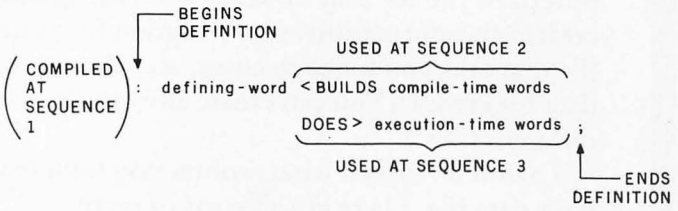


Figure 4: *The structure of the source definition of a defining word. These source lines create a defining word for a new family (sequence 1). Execution of the defining word (sequence 2) <BUILDS a dictionary definition for a new family member. The contents of that definition is constructed by the compile-time words. Executing any family member (sequence 3) DOES> (ie: executes) the execution-time words.*

The following words store a 5 into the variable.

5 %INTEREST !

Since VARIABLE is system-supplied, the sequence 1 event (the compilation of VARIABLE) occurred when the FORTH system was generated.

<BUILDS and DOES>

To illustrate a simple sequence 1 event, a definition of VARIABLE is presented.

: VARIABLE <BUILDS 2 ALLOT DOES> ;

The defining word { : } (colon) is used to compile the source definition of VARIABLE . To the word { : }, VARIABLE is an ordinary definition (level I), and its definition is a sequence 2 event for { : }. VARIABLE is a defining word because the special words <BUILDS and DOES> are used. (The < and > characters are part of the names of the words; they are used like parentheses to indicate that <BUILDS comes before DOES> .)

As illustrated in figure 4, a defining word specifies both the compile-time behavior (sequence 2) and the execution-time behavior (sequence 3) of all words compiled by this defining word. The sequence 2 behavior is specified by <BUIILDS and any following words up to DOES> . The sequence 3 behavior is specified by DOES> and any following words up to { ; }. The English meaning of <BUILDS is "compiles" and the meaning of DOES> is "executes."

Figure 5 demonstrates what occurs when VARIABLE is executed. The end result of the execution of VARIABLE is that a new dictionary definition is created for the word %INTEREST . The following describes each step in the compilation of %INTEREST :

1. The execution of VARIABLE causes <BUILDS to be executed. <BUILDS reads the next word-name after the word VARIABLE from the input text stream. (In this example, the next word-name is %INTEREST .)
2. <BUILDS then adds the head of a new definition to the end of the dictionary. Within this head, the name field contains the member's word-name

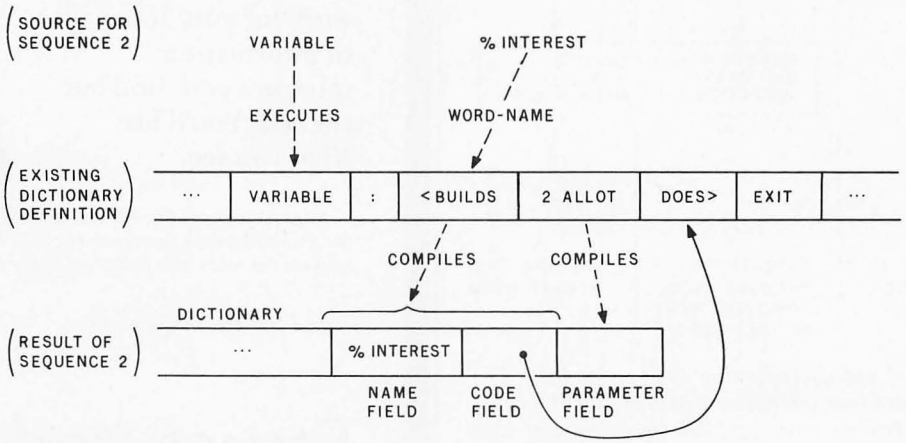


Figure 5: *The result of executing a defining word. The first line is executed, resulting in the compilation of the word-name %INTEREST . The result is a new definition in the dictionary.*

Now there's a new generation language that can teach your old TRS-80* some new tricks.

TFORTH.

A unique growth programming language for the TRS-80* that combines the best features of an interpreter and a compiler all in one functional easy-to-use package.

Introducing TFORTH from Sirius. A new and unique language that cannot be simply compared with FORTRAN or BASIC. TFORTH serves as an operating system, compiler, assembler, interpreter, virtual memory, file system, etc. all in one. Using concepts like virtual memory and stack organization, TFORTH makes easy, efficient, structures, re-entrant programs a natural consequence.

TFORTH is a procedural language specifying a process rather than a desired result. The ability to have the language grow in the direction the user desires is unusual but excellent for novel applications. New data types and new processes can become part of the language. Due to the modular construction, very compact code is produced which, even so, executes at speeds between machine code speeds and about 20% typical overhead. Memory requirements may be "less" than assembler coding or other high level languages.

TFORTH provides an excellent way to develop new languages, or provide simple control of devices including video monitors, A to D converters, burglar alarms and many other tasks requiring monitoring and decision. Many words to handle peripherals are part of basic TFORTH and others may be added easily. Often, substantial hardware development can be eliminated by using TFORTH to do the major digital logic or reduction of data.

The key to TFORTH's flexibility and ease of use lies in its use of a stack for parameters and a dictionary for WORDS. These WORDS are stated in terms of other WORDS already defined in the dictionary. To execute a "program", the WORD is typed on the console and that WORD is executed.

A rich set of WORDS comes with the basic language, providing IF-THEN-ELSE, DO LOOPS, BEGIN-END, virtual memory, any number base (to base 32) for input or output, a macro assembler, re-entrant code, multithread dictionary, line editor, excellent math package (16 bit integers, Double Precision Floating Point math [standard], SIN, COS, TAN, SQRT, EXP and LOG) and it runs under either TRSDOS* or NEWDOS. Assembler can be nested with high level in an easy fashion. Complicated drivers for new devices take only a few lines of TFORTH!

TFORTH from Sirius comes complete for the TRS-80* with as little as 16K of memory and a single Disk Drive using either TRS-DOS or NEWDOS. It is provided on diskettes and an optional Math and Utilities package is available.

TFORTH (on diskette--specify for Standard or 80 Track Drives). . . **$129.95**

TM Tandy Corp.

Among the many supported features of TFORTH are:

- **A Macro-Assembler**
- **Line Editor**
- **Advanced Math Package**
- **Re-Entrant Code**
- **Faster 1/0 Operation**
- **Super Graphics Capabilities**
- **High-Speed File Handling**

. . . and much more!

SIRIUS SYSTEMS

7528 Oak Ridge Highway
Knoxville, TN 37921 (615) 693-6583

Circle 116 on inquiry card.

(%INTEREST), and the code field contains a pointer to the instructions that will be executed when %INTEREST is executed (during sequence 3).

3. The two words { 2 ALLOT } are executed next. These will reserve 2 bytes of dictionary space for the value of the variable. This space is in the parameter field of the dictionary definition.
4. Finally, DOES> terminates the compilation of %INTEREST and links the code field of %INTEREST to the execution-time part of VARIABLE .

When %INTEREST is executed (sequence 3), DOES> is executed, followed by the FORTH words between DOES> and the end of the definition. (In this example, there are no words following DOES> ; the word EXIT is a routine left by the end-of-definition word { ; }.) DOES> returns the memory address of the parameter field within the dictionary definition of %INTEREST . Since the parameter field of a word defined by VARIABLE contains only the value of that word, execution of the word %INTEREST returns the *address* of its value, which is then pushed onto the parameter stack. (That is, in fact, the execution-time behavior of a FORTH variable.)

Figure 6 shows an example of the execution of %INTEREST .

[*The above definition and usage of the word* VARIABLE *are valid for existing* FORTHs. *However, the definition of* VARIABLE *supplied with most* FORTHs *requires the initial value of the variable before the word* VARIABLE (*eg*: { 5 VARIABLE %INTEREST }). *This definition of* VARIABLE *is:*

{ : VARIABLE <BUILDS , DOES> ; }

....GW]

The previous example demonstrated the following principles:

- Sequence 1: the definition of a defining word specifies both the compile-time behavior and execution-time behavior of all words belonging to the family of the defining word (ie: all words created using the defining word).
- Sequence 2: the execution of a defining word causes the compilation of the word-name(s) that follow. This creates a new dictionary

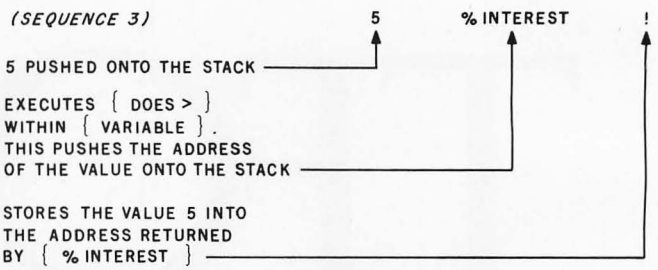
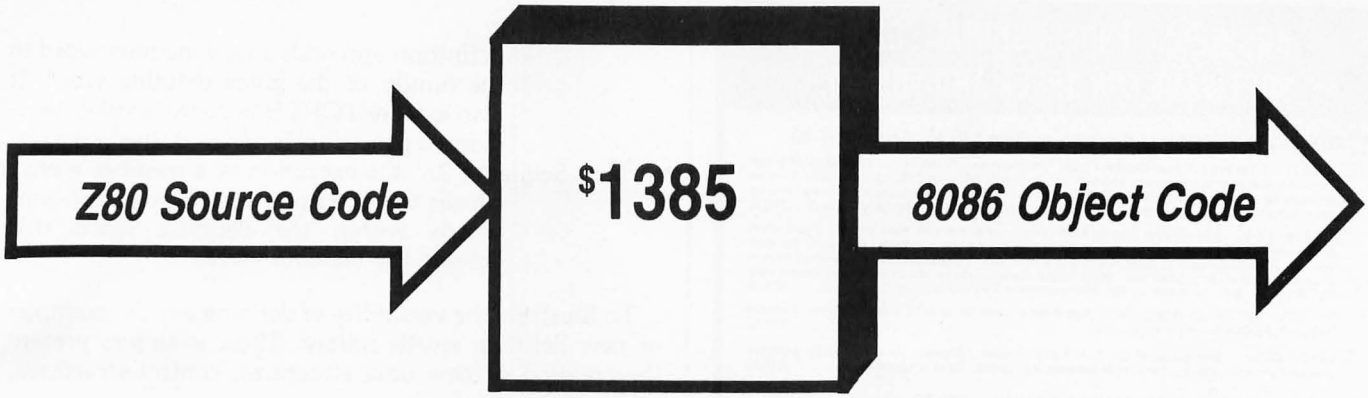


Figure 6: *The execution of a family member word. The value 5 is stored in the variable %INTEREST .*



86-DOS™ 8086 OPERATING SYSTEM - $95

1. Read Z80 source code file written in CP/M* format and convert to 86-DOS format.

2. Translator program translates Z80 source code to 8086 source code.

3. Resident assembler assembles the translated 8086 source code to 8086 object code.

4. Minor hand correction and optimization.
(A recent 19K Z80 program translation took us about four hours to fix up. Even without optimization, it ran twice as fast as the original!)

86-DOS™

This operating system is the first complete package of resident software for the 8086 offered in the general marketplace. 86-DOS provides a high-level interface between the program and its hardware environment, with functions such as console I/O of characters and strings, and random and sequential reading and writing of named disk files.

A multi-level hierarchy, made possible by the hardware address relocation inherent in the 8086, allows programs to run other programs. Complete error trapping of otherwise "fatal" errors (e.g. - disk hard errors) is possible allowing a running program to remain in control or to transfer control to a high level in the hierarchy.

The package includes an 8086 resident assembler, a Z80 to 8086 source code translator, a utility to read files written in CP/M format and convert them to 86-DOS format, a line editor, and disk maintenance utilities. Price (registered SCP CPU owners) - $95. Others - $195.

THE REMAINING $1290 BUYS OUR 8 MHZ. 16-BIT

8086 2-CARD CPU SET TO RUN IT ON

Also requires S-100 mainframe, 8-bit or 16-bit static memory (yours or ours), disk subsystem.

Seattle Computer Products, Inc.
1114 Industry Drive, Seattle, WA. 98188
(206) 575-1830

* CP/M is a registered trademark of Digital Research Corp.

Circle 119 on inquiry card.

definition and adds a new member word to the family of the given defining word. It also extends FORTH because another user-defined procedure is added to the language.

- Sequence 3: the execution of a member word causes the execution of the execution-time words within the defining word that created the member word.

To illustrate the versatility of defining words, examples of new defining words follow. These examples present the creation of new data structures, control structures, and software tools.

Creating a String-Handling Defining Word

To show how defining words can create data structures, a one-dimensional array of 8-bit values will be created. A defining word named STRING will be constructed. After STRING has been compiled, any number of strings may be created; each can have a different name and size. Before the definition for STRING is shown, an example will first be used to describe how STRING will be used.

To create a string 5 bytes long with the name BEANS , the following words would be used (BEANS is the name of the string, not the value put into the string):

```
5 STRING BEANS
```

This is a sequence 2 event that will create a dictionary definition for BEANS ; this definition will contain 5 bytes of data space for the value of the string.

To fetch or store a character in BEANS , a subscript will be passed to BEANS . BEANS will return the address of the subscripted byte. For example, the words

```
3 BEANS C@
```

would fetch character number 3 from BEANS . This is a sequence 3 event because it is a normal use of a word defined by STRING . The subscript precedes BEANS because FORTH prefers to pass data values on a stack.

The definition of STRING can now be written as shown in listing 1. This definition is similar to that of VARIABLE .

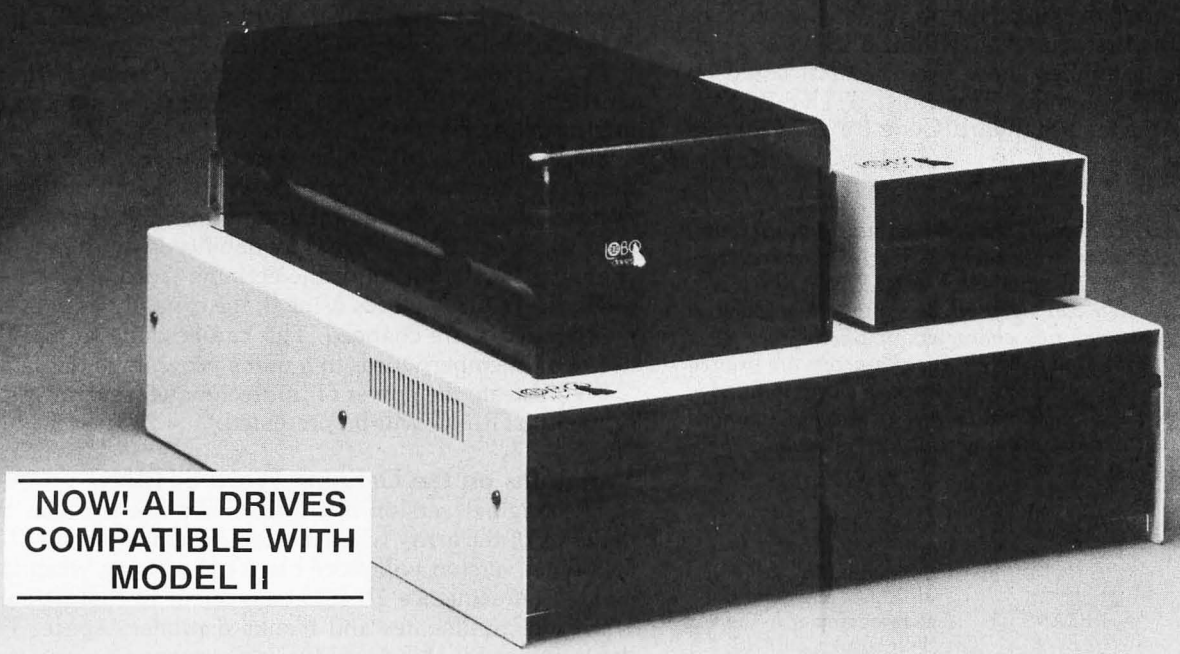
The parameter for ALLOT is omitted in this definition; the string size declaration at sequence 2 will supply the size parameter for ALLOT . (The word ALLOT looks for the number of bytes to be reserved to already be on the stack; this is why the string size precedes the word STRING when the string variable BEANS is defined.)

Following DOES> is the word + . This will add the address of the start of the string (supplied by DOES>) to the subscript (supplied to BEANS at sequence 3). Figure 7 illustrates how this works.

Listing 1: *A user-defined defining word. The word STRING , once defined, can be used to define new FORTH words with unique properties.*

```
                                        used at sequence 2 used at sequence 3
( defined at )
( sequence 1 ) : STRING  <BUILDS ALLOT    DOES> + ;
```

NEW FROM LOBO:



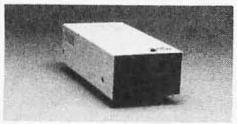
NOW! ALL DRIVES COMPATIBLE WITH MODEL II

An Entire Family of Disk Drives for APPLE, TRS-80*, and S-100 Computers

Only LOBO DRIVES offers you an entire family of fully-compatible disk drives to select from. Whatever computer you're using, APPLE, TRS-80, or S-100, you can add a LOBO drive now, with the peace-of-mind of knowing there's a whole family of drives available when you're ready to expand.

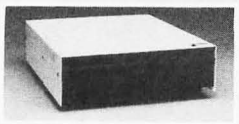
And every drive you order comes complete with chassis and high reliability power supply. Each drive is 100% calibrated, burned-in, and performance tested on either an APPLE, TRS-80, or S-100 computer before it's shipped. We are so proud of our drives . . . our quality, reliability, and performance, that we back-up every drive with a one year, 100% parts/labor warranty.

400 SERIES FLOPPY DISK DRIVES



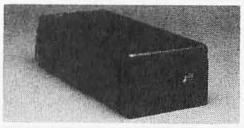
Meet our low-cost 5.25-inch mini drive that records data in either hard or soft sectored format. It is available in single or double density configurations, with a total storage capacity of 220K bytes.

800/801 SERIES FLOPPY DISK DRIVES



Here is our dual 8-inch Floppy disk memory unit. It records and retrieves data on standard 8-inch diskettes to provide 800K bytes of data storage unformatted, or 512K bytes in IBM format per drive. It is also available with double-sided, double-density capabilities, for a maximum storage capacity of 1.6 Megabytes.

7000 SERIES HARD DISK DRIVES



The latest member of our drive family, the Series 7000 is an 8-inch, 10 Megabyte Winchester Technology, hard disk drive. It is fully hardware/software compatible and comes complete with disk controller. Now you can have the convenience, speed, reliability, and all the storage capacity you need.

Call or write for the complete LOBO DRIVES story. Find out just how competitively priced a quality drive can be.

Quantity discounts available –
Dealer inquiries invited.

Yes, I want to know more about LOBO Drives and what they can do. Send me information on:
☐ TRS-80 ☐ APPLE ☐ S-100

☐ 5 1/4-in. Floppy drive
☐ 8-in. Floppy drive
Single sided
Double sided
☐ 8-in. Winchester hard disk, 10 Mbyte drive
☐ Double density expansion interface

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone No. _____

If dealer, provide resale no. _____

LOBO drives INTERNATIONAL

935 Camino Del Sur
Goleta, California 93017
(805) 685-4546

"CAN YOU REALLY AFFORD TO PAY LESS?"

*TRS-80 is a registered trademark of Radio Shack, a Tandy Company.

When STRING is executed (sequence 2), it builds a dictionary definition for BEANS , which is allotted 5 bytes of data space. When BEANS is executed (sequence 3), it does the addition of the subscript on top of the stack to the address of the first character within BEANS .

The following examples show how BEANS could be used in a FORTH program. The word STUFF-BEANS will store the American Standard Code for Information Interchange (ASCII) characters A thru E in the string variable BEANS . (See listing 2a.) The word SPILL-BEANS will print the characters in BEANS on the user's terminal. (See listing 2b.) Using these words would produce the results shown in listing 2c.

In a similar way, multidimensional-array defining words may be defined; the size of each element can be any number of bytes.

Since the execution-time function of all family members is specified only once in the definition of the family's defining word, programming time is reduced, memory space is saved, and readability is improved. By changing the definition of the defining word and recompiling the FORTH words using it, the capabilities of every member word are changed. This can be done so that the use of all member words in a user's program is the same.

To illustrate the power of this technique, several variations on STRING will be presented.

Listing 2: *Using a FORTH word created by a user-defined defining word. The 5-character string variable BEANS was previously defined with the FORTH statement { 5 STRING BEANS }. Now the word BEANS can be used like any other word in FORTH. In listing 2a, the five characters of BEANS are filled with the letters A thru E. In listing 2b, the characters are printed out. Listing 2c gives the results of executing the words defined in listings 2a and 2b. (The underline denotes user input followed by a carriage return; the computer output, not underlined, follows.)*

(a)

```
: STUFF-BEANS   5 0 DO          ( for all of 'BEANS' )
                   I 65 +       ( add 65 decimal to )
                                ( do-loop index, yielding )
                                ( an ASCII character )
                   I BEANS C!   ( store character in the )
                                ( 'I' th byte of 'BEANS' )
                LOOP
;
```

(b)

```
:SPILL-BEANS    5 0 DO          ( for all of 'BEANS' )
                   I BEANS C@   ( fetch the 'I' th character )
                   EMIT         ( print it )
                LOOP SPACE      ( print an extra space )
;
```

(c)

```
STUFF-BEANS   OK
SPILL-BEANS   ABCDE OK
```

Variations on the Defining Word STRING

The original version of STRING did not initialize the contents of the array when it created member arrays. The following version will store blanks in a string when it is created (at sequence 2). It is convenient to first define a word which allocates and blanks dictionary space. The definition of BLANK&ALLOT is a sequence 2 event. (See listing 3a.)

Next, we create a new version of STRING that is the same as the original, except that BLANK&ALLOT is substituted for ALLOT . (See listing 3b.) (The redefinition of STRING is a sequence 1 event.) This version is used exactly like the original, but initialized strings are created automatically.

Another variation of STRING checks if a subscript exceeds the string size when member strings are executed (at sequence 3). If the subscript is less than the string size, the result is the same as before; but, if the subscript is negative or greater than the string size, an error message

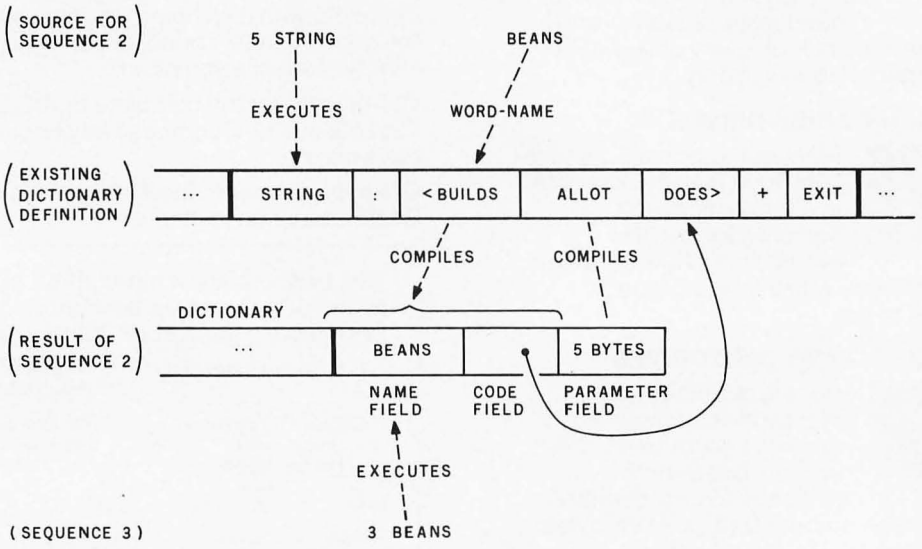


Figure 7: *The creation and use of a character array. The defining word STRING is executed, causing the compilation of a dictionary definition for BEANS containing 5 bytes of data space. When BEANS is executed (last line), the DOES> part of the definition of STRING adds the address of the parameter field of BEANS to the subscript (which is 3), returning the address of the desired character within BEANS .*

Listing 3: *A more sophisticated definition of STRING . The word BLANK&ALLOT (shown in listing 3a) allocates space for and assigns blanks to a newly defined string. The new definition of STRING (shown in listing 3b) uses BLANK&ALLOT to blank out a string when it is created.*

```
: BLANK&ALLOT
          HERE                ( get the address of the )
                              ( start of the string )
(a)       OVER BLANK          ( store blanks in the string )
          ALLOT               ( allocate space for the array )
;


: STRING   <BUILDS   BLANK&ALLOT  ( used at sequence 2 )
(b)        DOES> +                ( used at sequence 3 )
;
```

Listing 4: *Another definition of STRING . This definition stores the size of the string variable when the variable is created and checks for a correct subscript when a character within the string variable is referenced.*

```
: STRING
        ( used at sequence 2: )
        <BUILDS DUP  ,          ( store string size in )
                                ( member's parameter field )
           ALLOT                ( allocate string space )
        ( used at sequence 3: )
        DOES>   2DUP            ( duplicate both the subscript )
                                ( & parameter field address )
           @ U< IF              ( if the subscript is less )
                                ( than the string size )
              +                 ( add subscript to address )
              2+                ( step over the string size )
                                ( stored in the first 2 bytes )
           ELSE                 ( otherwise the subscript )
                                ( is too large or negative )
           ." RANGE ERROR"      ( print error message )
           OVER . @ .           ( print string size and )
                                ( and bad subscript )
           2+                   ( leave address of first byte, )
                                ( a "safe" address )
           THEN
;
```

is produced and the illegal subscript is printed. The string size must be stored in the dictionary definition of member strings when they are compiled (at sequence 2) so that the range check can be made when they are executed (at sequence 3).

A new definition of STRING (a sequence 1 event) that does the subscript checking defined previously is given in listing 4.

The range check slows the execution of every reference to a member string, but such checking may be useful during program development. Since this version and the original version defining STRING are used exactly the same, it is possible to compile this definition of STRING while debugging (then compile all references to it or its member strings). After the program has been debugged, the original version can be compiled (followed by the compilation of all references to it or its members), and the program will run faster.

The next version of STRING allows very large strings to be created and used.

Virtual Strings in FORTH

If the maximum string size exceeds the amount of programmable memory in the computer, the only solution is to write your program using *virtual memory management*. This means that data stored on disk or tape is considered part of the memory of the computer, and that all operations working on these data take care of reading and writing data between main memory and the magnetic storage device.

Using virtual memory management, a program can operate on a string array that is larger than main memory; pieces of the string can be read into memory and written back to disk or tape when required. And, although this technique will slow the execution rate of a program using it, it may be the only way to get a problem solved—and better a slow solution than none at all.

(It is more common to need to manipulate large arrays of numbers rather than strings. Still, the same technique described here can be applied to numeric or any other kind of array.)

With most traditional languages, it would be necessary to rewrite the user program so that all array references would call some function that could perform the disk read operations. Execution time could be decreased if frequently referenced array elements were kept in memory as much as possible. Therefore, it would help if our virtual-memory-array program could keep track of what data is in memory as the program executes.

To show the difficulty of implementing this technique in traditional languages, a FORTRAN example will be used. In standard FORTRAN, the statement:

```
ARRAY(5,7,2)=AR1(1,2)+AR2(10,20,30)
```

is equivalent to the FORTH words:

```
1 2 AR1 @     10 20 30 AR2 @     +
5 7 2 ARRAY   !
```

In either FORTRAN or FORTH, if the arrays could not fit into memory and were instead on disk, the array references would have to be changed so that some additional procedures read and wrote selected pieces of data between disk and memory. But in FORTRAN, the entire source program would have to be changed. (In FORTH, the body of the program would remain the same; only the appropriate defining word would be changed.)

The following might be the simplest modification possible in standard FORTRAN to do the previous statement using virtual memory management of the arrays:

```
TEMP=FETCH2(AR1(1,1), 1,2)+FETCH3(AR2(1,1,1),
     10,20,30)
CALL STORE3(ARRAY(1,1,1), 5,7,2, TEMP)
```

The functions FETCH2 and FETCH3 are user-written procedures to read the referenced array elements. The subroutine STORE3 is a user-written procedure to write a given value into an assigned array element. If a large program using many normal array references had to be changed to use FETCH and STORE calls, a *lot* of work would be required.

FORTH's separation of control between defining words and their members permits the necessary changes to be made in the definition of the defining word; in this

SAVE / on Software and Hardware for TRS-80® and Apple®

NEWDOS/80

A new enhanced NEWDOS for the TRS-80.

The most powerful Disk Operating System for the TRS-80 designed for the sophisticated user and professional programmer who demands the ultimate.

NEWDOS/80 is the planned upgrade from NEWDOS 2.1. Some of the features are:

- New BASIC commands for files with variable record lengths up to 4095.
- Mix or match drives. Use 35, 40 or 77 track 5" disk drives or 8" disk drives, or combo.
- Security boot-up for BASIC or machine code application programs.
- New editing commands.
- Enhanced RENUMber that allows relocation.
- Command chaining.
- Print Spooler.
- DFG function; striking of D, F and G keys allows user to enter a mini-DOS without disturbing program.
- Compatible with NEWDOS & TRSDOS.
- Machine language Superzap/80, 2.1 utilities and enhanced copy by file commands.
- Enter debug any time by pressing 123 keys. Also allows disk I/O.
- Diskette "Purge" command.
- Specifiable system options (limited sysgen type commands).
- Increased directory capacity.

$149

APEX

A new disk operating system for the Apple.

Fully Professional DOS for the Apple II. The result of two years of extensive development, APEX provides a complete program development and file management system, both powerful and useable. A comprehensive command set allows the user to perform almost any imaginable disk operation. Here are some of APEX's features:

- Command structure similar to CPM and main frame systems. Contains 20 command words, with ability to treat external programs as transient commands to the operating system.
- Scrolling editor compatible with Videx 80 char. card.
- Easy program interface. Simple communications between the DOS and user program.
- Capable of handling 5 inch, 8 inch and hard disks.
- Safety features to protect against accidental data loss. Features include backup files, directory, read-after-write and limit checks.
- 4 times faster than CPM.
- Auto default structure eliminates tedious typing by automatically setting up command strings, file names, etc.
- Functional on both single and multi-drive systems. Includes ultilities for file copy.
- Device handler structure for interfacing peripherals.

The APEX package includes all of the tools for a complete assembly language development system, high speed two pass resident assembler and a powerful macro editor.

The complete APEX package with operating system, assembler, editor and manuals also includes utilities to maintain files on single or multiple drive systems.

$99

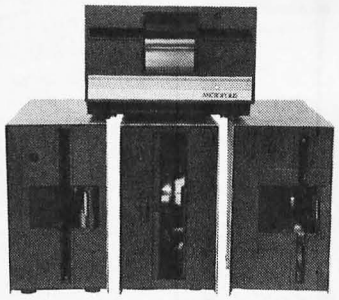
Related Software
XPLØ $79
FOCAL™ $59

Disk Drive Sale!

$70 worth of FREE merchandise with purchase of Shugart SA400 with power supply and chassis, the disk that Radio Shack sells for $499.

SAVE $200. ..... $369
TF-Pertec FD200, 40 track, use both sides. $389
TF-5 MPI B51, 40 track. $389
TF-70 Micropolis, 77 track with 195K storage. $639
TDH-1 Dual Sided drive, 35 track. $499

NEWDOS+, 40 track $110
35 track ..... $ 99
Business Programs (Interactive A/R, A/P, & GL) $349
Radex Data Base Program ..... $ 99
Mailing List $59



Disk Drive Expansion System

	List Price
• 2 Shugart SA400 drives with power/chassis	$ 738
• 1 Two-Drive Cable	$ 25
• 1 Expansion Interface 32K	$ 499
• 1 35-track DOS+	$ 99
TOTAL LIST PRICE	$1,361
SPECIAL PRICE ONLY	**$1,199**

MOD I 8" Disk System
• One SA800R Floppy
• 2 Drive Chassis and Power Supply
• DOS and Cable **$1,095**

MOD II 8" Disk System
• 3 Drive Chassis
• 2 Drive Expansion System **$1,399**
• 3rd Drive ... **Add $479**

Drives for any microcomputer

Does not include power supply & cabinet.

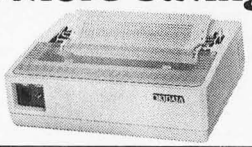
Pertec FD200	$ 282	FD250	$ 399
Shugart SA400	$ 279	SA800/801	$ 479
MPI B51	$ 279	B52	$ 349

PRINTERS

Centronics 779 .. $1,069
Base 2 ..... $ 599
Centronics 737
Text processing capabilities, lower case descenders, underlining, subscripts and superscripts, 80 cps $899

Spinwriter $2,549

More Savings



NEW SALE PRICE
TRS-80 Graphics, Okidata Microline 80...
List $949
Our price **$699**

Save on Apple II 16K
FREE memory upgrade kit to 48K with purchase of Apple II 16K **$1,195**

Introductory Offer — Mini-Floppy for Apple (2nd Drive) ... Only **$419**

Apple 8" Disk System • One SA800R Floppy
• 2 Drive Chassis & Power Supply
• Controller, Cable and DOS **$1,449**

16K Memory Upgrade Kits ..... $ 79

0/2A

VISA master charge

Apparat, Inc.

(303) 741-1778
4401 S. Tamarac Pkwy. • Denver, CO 80237 • (303) 758-7275

MTI MICROCOMPUTER TECHNOLOGY INCORPORATED

3304 W. MacArthur • Santa Ana, CA 92704 • (714) 979-9923

All prices cash discounted / Freight: FOB factory. Ask for our free catalog.

Circle 139 for APPARAT

Circle 125 for MTI

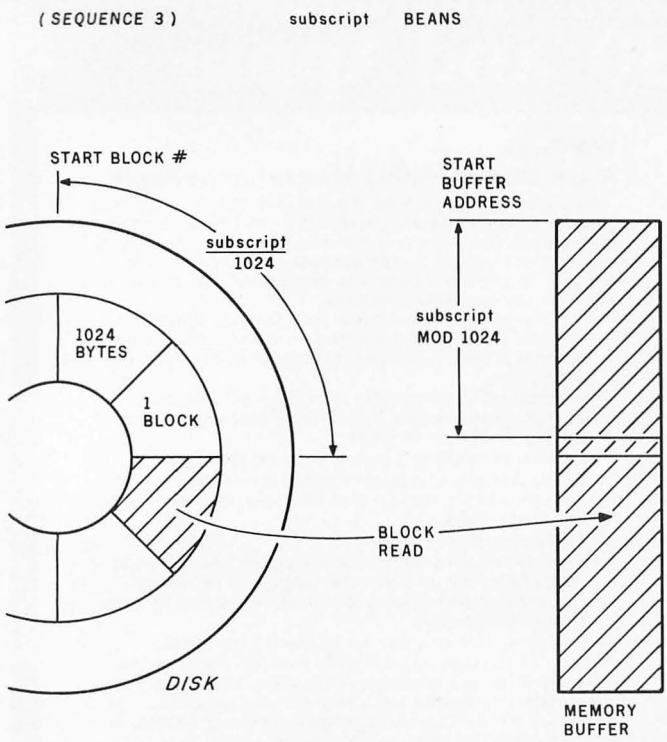


Figure 8: *Accessing a virtual array. The data for a large array is kept on a disk. When a byte is referenced, BEANS is executed. One block containing the byte is read into a memory buffer (if it is not already present). Finally, the memory address of the referenced byte is returned by BEANS .*

way, the program that uses the arrays does not have to be changed.

Furthermore, FORTH's virtual memory facility for disk reading and writing automatically keeps track of what data has been read into memory and tries to keep frequently referenced sections in memory.

Figure 8 illustrates how the array will be read in blocks of 1024 bytes into memory buffers. The new definition for the defining word STRING is given in listing 5.

Adding New Control Structures with Defining Words

The next example illustrates the use of defining words to add control structures to the FORTH compiler. FORTH supplies { IF ... ELSE ... THEN } compiler structures and also loop structures like { DO ... LOOP },{ BEGIN ... UNTIL }, *and* { BEGIN ... WHILE ... REPEAT } loops.

In this example, we will create a *case* (choose one of *n* alternatives) selection mechanism. A *case number* will designate one of several words to be executed. Figure 9 presents how a case statement selects one of several procedures for execution. No matter which one is chosen, execution continues with one common procedure that follows the case structure.

The new defining word will be named { CASE: } and can be used similarly to { : }, as the following

Listing 5: *Another definition of STRING . This definition creates a virtual string array that stores the string on disk and reads it into main memory when necessary. With this definition of STRING , it is possible to manipulate a string that is larger than main memory without changing the program that uses the long string. The disk operations are transparent—that is, the programmer does not know he is using the disk except for response time.*

```
: STRING
        ( used at sequence 2 )
        <BUILDS   NEXT-BLOCK#( get the next available )
                              ( disk block # )
                  ,           ( store it in the member's )
                              ( parameter field )
                  DISK-ALLOT  ( reserve disk space for )
                              ( the array )
        ( used at sequence 3 )
        DOES>     @           ( get start-block # )
                  SWAP        ( subscript on top, )
                              ( start-block # beneath )
                  1024  /MOD  ( divide subscript by )
                              ( # bytes in a disk block; )
                              ( the quotient is the block )
                              ( index within the array; )
                              ( the remainder is the byte )
                              ( index within the block )
                  ROT   +     ( add start-block # to the )
                              ( block index )
                  BLOCK       ( call the FORTH virtual )
                              ( disk manager to read the )
                              ( referenced block; )
                              ( if it is already in memory )
                              ( no read is performed )
                  +           ( add the byte index to the )
                              ( memory address of the )
                              ( buffer where the block is )
                              ( located, the result is )
                              ( a memory address of the )
                              ( byte specified by the )
                              ( subscript before BEANS )
;
```

CompuPro S-100 Motherboards:
Designed for the Future,
AVAILABLE NOW

You won't have to throw away these motherboards when you upgrade your system – they are specifically designed to handle the new generation of 5 to 10 MHz CPUs coming on line, as well as present day 2 and 4 MHz systems. Faraday shielding between all bus signal lines minimizes crosstalk; additionally, when signal lines cross each other on opposite sides of the board, they do so at a 90 degree angle to minimize any chance of stray coupling. You'd expect the company that pioneered active termination to include true active termination, but we've gone one step better by splitting the termination load between each end of every bus line. And you won't have to junk your present computer box with our new motherboards – all sizes fit Godbout, Vector, Imsai, TEI, and similar enclosures.

These high-performance motherboards are available in **"unkit"** form (edge connectors and termination resistors pre-soldered in place for easy assembly), or fully **assembled** and ready to go.

#CK-024 20 slot motherboard with edge connectors – unkit $174, assm $214
#CK-025 12 slot motherboard with edge connectors – unkit $129, assm $169
#CK-026 6 slot motherboard with edge connectors – unkit $89, assm $129

NOTE: Most **CompuPro** boards are available in **unkit** form (sockets, bypass caps pre-soldered in place), **assembled,** or qualified under the **Certified System Component** (CSC) high-reliability program (200 hour burn-in, more). CSC memory boards run at 8 MHz, are **guaranteed** to run with 6 MHz Z-80s, and draw even less power than standard models.

CAREFUL . . . NOT ALL S-100 CPU BOARDS ARE CREATED EQUAL!

You'll appreciate the extras that go into our CPU boards; take IEEE spec compatibility, for example. While others may claim compatibility, we meet **all** timing specs – and we'll be glad to send you timing diagrams for our CPUs to prove it (just include an SASE). You don't have to compromise on another "me-too" board . . . choose **CompuPro.**

THE ENHANCED/ADVANCED Z-80A S-100 CPU BOARD

Superior design in an IEEE-compatible board gives the power for future expansion as well as system flexibility. Includes all standard Z-80A features along with power on jump/clear, on-board fully maskable interrupts for interrupt-driven systems, selectable automatic wait state insertion, provision for adding up to 8K of on-board EPROM, 4 MHz operation, and IEEE compatible 16/24 bit extended addressing. **$225 unkit, $295 assm, $395 CSC.**

THE COMPUPRO "RAM" SERIES OF STATIC MEMORY

Recommended for commercial, industrial, and scientific applications. 4/5 MHz standard operation, no dynamic timing problems, meets all IEEE specifications, low power/high speed chips used throughout, extensive bypassing, careful thermal design.

S-100 STANDARD MEMORY	unkit	assm	CSC
8K RAM IIA	$169	$189	$239
16K RAM X-16	$329	$379	$479
24K RAM XX-24	$449	$499	$599
32K RAM X-32	$599	$689	$789
S-100 EXTENDED ADDRESSING MEMORY (16/24 address lines; addressable on 4K boundaries)			
16K RAM XIV	$299	$349	$429
S-100 BANK SELECT MEMORY (Cromemco etc. compatible; addressable on 4K boundaries)			
16K RAM XIIIA-16	$349	$419	$519
24K RAM XIIIA-24	$479	$539	$649
32K RAM XIIIA-32	$649	$729	$849
SBC/BLC MEMORY			
32K RAM XI	n/a	n/a	$1050

OTHER S-100 BUS PRODUCTS

Godbout Computer Enclosure . . . . . **$289 desktop, $329 rack mount**
Active Terminator Board . . . . . **$34.50 kit**
2708 EPROM Board (less EPROMs) . . . . . **$85 unkit**
Memory Manager Board . . . . . **$59 unkit, $85 assm, $100 CSC**
2S "Interfacer I" I/O Board . . . . . **$199 unkit, $249 assm, $324 CSC**
3P Plus S "Interfacer II" I/O Board . . . . . **$199 unkit, $249 assm, $324 CSC**
Mullen Extender Board . . . . . **$59 kit**
Mullen Relay/Opto-Isolator Control Board . . **$129 kit, $179 assm**
Vector 8800V S-100 Prototyping Board **$19.95**

TERMS: Cal res add tax. Allow 5% for shipping, excess refunded. VISA®/Mastercard® orders ($25 min) call **(415) 562-0636**, 24 hrs. COD OK with street address for UPS. Prices good through cover month of magazine.

NEW! S-100 DUAL PROCESSOR CPU BOARD

The **Dual Processor Board** is here . . . and CPU boards will never be the same again. 8088 CPU gives true 16 bit power with a standard 8 bit S-100 bus; an 8085 gives compatibility with CP/M and 8080 software. Accesses up to 16 megabytes of memory, meets all IEEE S-100 bus specifications, runs 8085 and 8086 code **in your existing mainframe** as well as Microsoft 8086 BASIC and Sorcim PASCAL/M™, runs at 5 MHz for speed as well as power, and is built to the same stringent standards that have established our leadership in S-100 bus components. **Introductory prices: $385 unkit, $495 assm, $595 CSC.**

8085 single processor version of above: **introductory prices $235 unkit, $325 assm, $425 CSC.**

SPECTRUM S-100 COLOR GRAPHICS BOARD

Includes 8K of IEEE-compatible static RAM; full duplex bidirectional parallel I/O port for keyboard, joystick, etc. interface; and 6847-based graphics generator that can display all 64 ASCII characters. 10 modes of operation, from alphanumeric/semi-graphics in 8 colors to ultra-dense 256 × 192 full graphics. 75 Ohm RS-170 line output and video output for use with FCC approved modulators. **Introductory prices: $339 unkit, $399 assm, $449 CSC.** Don't settle for black and white graphics or stripped-down color boards; specify the **CompuPro Spectrum.**

Want graphics software? **Sublogic's 2D Universal Graphics Interpreter** (normally $35) is yours for **$25** with any Spectrum board purchase.

16K DYNAMIC RAM SPECIAL: 8/$59!

Expand memory in **TRS-80* -I** and **-II**, as well as machines made by **Apple, Exidy, Heath H89,** newer **PETs,** etc. Low power, high speed (4 MHz). Add $3 for 2 dip shunts plus TRS-80* conversion instructions. Limited quantity. *TRS-80 is a trademark of the Tandy Corporation.

PASCAL/M™ + MEMORY SPECIAL

PASCAL – easy to learn, easy to apply – can give a microcomputer with CP/M more power than many minis. We supply a totally standard **Wirth PASCAL/M™ 8" diskette** by **Sorcim,** with manual, for **$150 with the purchase of any memory board.** Specify Z-80 or 8080/8085 version. PASCAL/M™ available separately for $175; manual available separately for $10.

COMING SOON!

We've got a new board coming up that's **so** versatile some of our people have nicknamed it the "smorgasboard": it includes (among other things) a real-time clock, interval timer, interrupt controllers, and math processor. We've also got a board in the works that greatly enhances the throughput and performance of multi-user (2 or more terminal) systems, by assuming a lot of the overhead functions normally handled by the main CPU. Look for more details on these useful and functional products in the months ahead, or check with finer computer stores for additional information on these and other **CompuPro** products.

CompuPro™ from **GODBOUT ELECTRONICS**

Bldg. 725, Oakland Airport, CA 94614

Circle 127 on inquiry card.

example shows. (In this implementation of the *case* construct, the selection of a case causes the execution of one FORTH word. Since there is no restriction as to the internal complexity of a given word, the selection of one case can cause any combination of conditional, loop, or case structures to be executed.)

In our example, let us first define three words, 0PET , 1PET , and 2PET , that are to be executed when the value on top of the stack is 0, 1, or 2, respectively. This is done in listing 6a. Then we use the { CASE: } defining word (which we will look at later) to define the word ANIMAL (listing 6b). Now that ANIMAL and the case words it uses are defined, calling ANIMAL with the appropriate value on the stack executes the proper case word (listing 6c). For example, pushing a 2 onto the stack and calling ANIMAL causes word 2PET to be executed; this causes the English word COUGAR to be printed.

Since { CASE: } is a defining word, ANIMAL is a member of the { CASE: } family. The definition of ANIMAL consists of a list of addresses for the case words associated with ANIMAL.

The definition of { CASE: } is a sequence 1 event. Listing 7 shows the definition of { CASE: } in FORTH-79. [*Listings 8a and 8b show the same definition for fig-FORTH and MMSFORTH, respectively....*GW] Figure 10 shows how the word ANIMAL is built using { CASE: }. The { : } compiler is used to compile the words following ANIMAL. When ANIMAL is

Listing 6: *Example of a new user-defined programming construct. In listing 6a, we define the words we want to execute when the numbers 0, 1, and 2 are on top of the parameter stack. In listing 6b, the user-defined defining word { CASE: } defines the word ANIMAL , which will execute 0PET , 1PET , or 2PET , depending on the value on top of the parameter stack. Listing 6c illustrates what happens when the case-word ANIMAL is executed. See listing 7 for the definition of { CASE: } .*

```
: 0PET    ." AARDVARK " ;          ( print the quoted string )
: 1PET    ." BEAVER " ;            ( when executed )
: 2PET    ." COUGAR " ;
```

(a)

```
( sequence 2 )      CASE: ANIMAL    0PET 1PET 2PET ;
```

(b)

```
( sequence 3 )      0 ANIMAL    AARDVARK OK
                    1 ANIMAL    BEAVER OK
                    2 ANIMAL    COUGAR OK
```

(c)

Listing 7: *Definition of the defining word { CASE: } in FORTH-79. This word allows the user to create case-words that execute one of several FORTH words depending on the value on top of the parameter stack.*

```
: CASE:
        ( used at sequence 2 )
          <BUILDS                ( create head for member )
              ]                  ( begin ':' compilation )
        ( used at sequence 3 )
          DOES>
            SWAP 2*              ( convert case number to )
                                 ( a byte index )
            + @                  ( fetch the address of the )
                                 ( indexed case word )
            EXECUTE              ( execute the selected word )
```

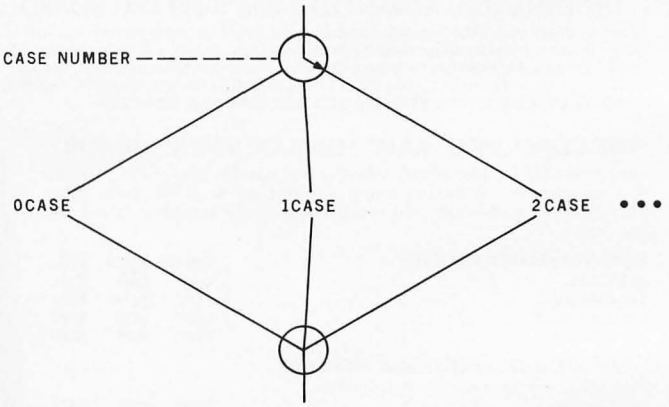


Figure 9: *The function of a* case *control structure. The* case number *selects one of several procedures for execution, then continues along a single exit path.*

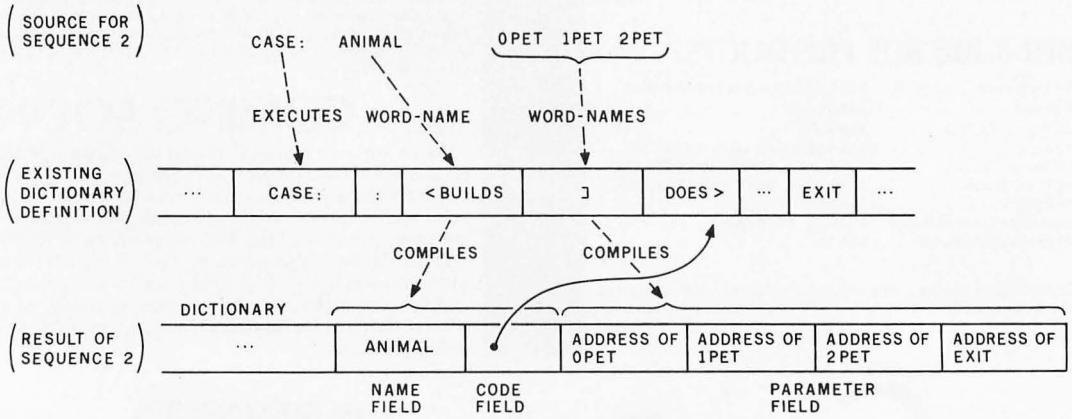


Figure 10: *The creation of a case control word. The execution of { CASE: } causes a definition for ANIMAL to be appended to the dictionary. The ']' word uses the { : } compiler to compile the addresses of the case words following ANIMAL .*

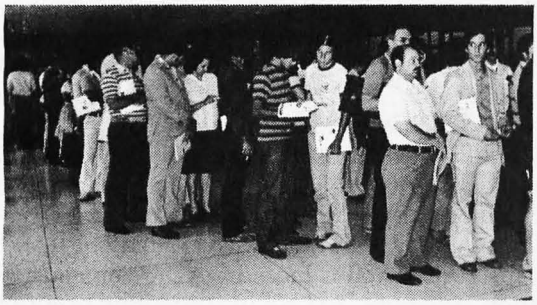
"THE ORIGINAL"

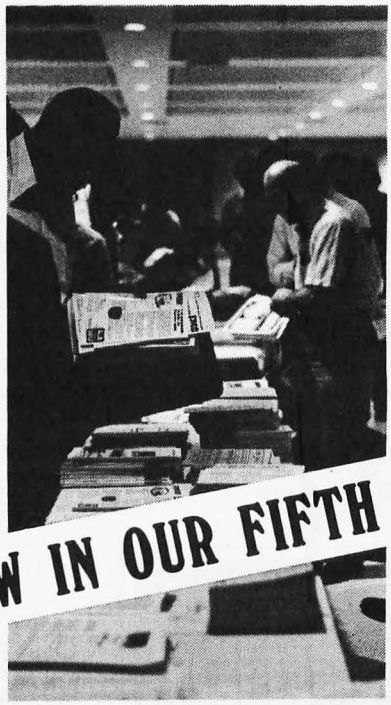
Personal Computing® 80

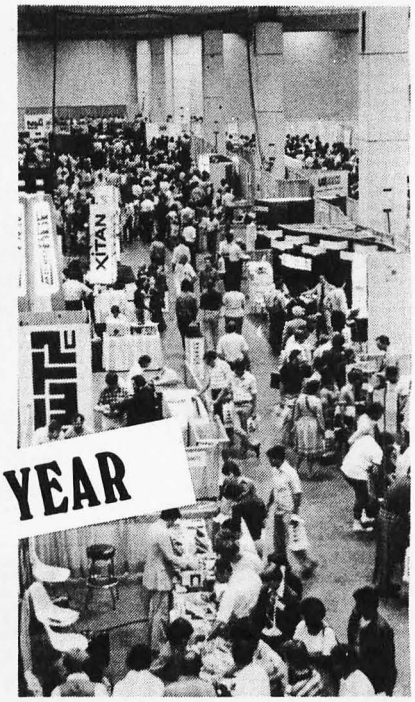
Presents:

Personal Computing and Small Business Computer Show

The Largest Personal Computing Show in 1980







NOW IN OUR FIFTH YEAR

August 21, 22,23, 24th at the Philadelphia Civic Center

- **Major exhibits by the industries leading companies**
- **Thursday, Aug. 21st, Dealer Day ——— 12 Noon to 6 P.M.**
- **Friday and Saturday, Aug. 22, 23rd ——— 9 A.M. to 6 P.M.**
- **Sunday, Aug. 24th ——— 10 A.M. to 5 P.M.**
- **Free Seminars • Robotics Contest • Antique Computer Display**
- **Special Seminars and Tutorials about Computer Music,** Saturday, Aug. 23rd
- **3rd Annual Computer Music Festival,** Saturday Evening, Aug. 23rd
  (Computer Music Festival is sponsored by the Philadelphia Area Computer Society-Tickets on sale at show)
- **Computer Visual Arts Festival,** Sunday, Aug. 24th

Advanced Registration Saves Time & Money

☐ Send_____Dealer-Retailer (4 days) Registrations at $10. each, $12. at door for Thursday-Sunday, Aug. 21, 22, 23, 24

☐ Send_____Regular Registrations (3 days) at $8. each, $10. at door for Friday-Sunday, Aug. 22, 23, 24 only.

Advanced Registrations will be mailed late July - early August. No Advanced Registrations accepted after Aug. 8th.

☐ Send Exhibitor information or Phone 609-653-1188

COMPANY NAME _____

NAME _____

STREET _____

CITY _____ STATE_____ ZIP_____

PHONE _____

Send To:

PERSONAL COMPUTING 80

Rt. 1, Box 242, Warf Rd., • Mays Landing, NJ 08330

Circle 128 on inquiry card.

Listing 8: *Definition of the defining word { CASE: } in fig-FORTH (listing 8a) and in MMSFORTH (listing 8b).*

```
( CASE:   as implemented in fig-FORTH)
: CASE:    <BUILDS    SMUDGE ]
           D O E S >  SWAP 2*
(a)                   + @
                      EXECUTE
;
```

```
(  CASE:    as implemented in MMSFORTH)
(  new word )) replaces  SMUDGE     )
: ))   1 STATE C! 21144 ;
: CASE    <BUILDS     ))
          DOES>       SWAP 2*
(b)                   + @ 2+
                      EXECUTE
;
```

Listing 9: *Definition of a defining word that acts as a programming tool. The word LOADED-BY allows the user to execute (or load) a screen by name rather than by number. For example, if you define { 125 LOADED-BY ACCOUNTING }, executing the word ACCOUNTING will have the same effect as executing the phrase { 125 LOAD }.*

```
( sequence 1 )  : LOADED-BY   <BUILDS ,    ( store screen # )
                                           ( in members def. )
                            DOES>   @      ( fetch screen # )
                                    LOAD   ( load it )
;
```

executed, the case number that precedes it (which is now on top of the stack) is used just like an array subscript to calculate the address of the case word to be executed. Its compiled address is then fetched and executed.

As with array-defining words, many variations of { CASE: } can be constructed. A case number-range check may be added. An "otherwise" case word can be specified to be executed whenever the case number is out of range.

Defining Words as Programming Tools

The final example applies defining words to the creation of software tools. Such tools are conveniences for the user. Good tools can increase a programmer's productivity, reduce errors, and improve program readability. Defining words can be used to add powerful tools to the FORTH language and operating system.

In FORTH, the word LOAD will compile source definitions from the disk starting at a specified screen number. A *screen* is a block of disk space where source text can be stored using an editor. Additional screens may be loaded if the initial screen contains more LOAD commands.

Application programs and utility programs begin on various screen numbers determined by the user. The defining word LOADED-BY allows words to be defined which will LOAD a screen without calling it by number.

For example, assume a business application starts on screen 125. Then the defining word LOADED-BY can be used to define a word that will load screen 125 when the member word is executed. When we define:

125 LOADED-BY ACCOUNTING

screen 125 will be loaded when the single word ACCOUNTING is executed. (If LOADED-BY *looks strange*, think of it as a FORTH word like VARIABLE .)

The definition of LOADED-BY is given in listing 9. This definition is similar to the definition of the word CONSTANT except that, rather than returning the value stored in the definition of the member word, LOADED-BY uses that value to provide a parameter to the word LOAD .

Summary

FORTH exploits its own extensibility to support a user's need for a variety of language facilities and compiler structures.

A defining word controls the compilation and execution of all words compiled by it. New defining words that define a new family of capabilities may be constructed. Subsequently, any number of individual members can be added to the family.

The source definitions of most defining words are short and simple. Proper use of defining words in a software development project reduces program development time, improves program readability, and makes program modification and maintenance easier.

Defining words are applicable to data structures, control structures used by the FORTH compiler, and software tools. The ability to create new kinds of defining words (which are, in their own way, small compilers) is a unique feature of FORTH and is one of the most powerful programming tools in the language. ■

Gregg Williams
Editor

This glossary is a compilation of most of the FORTH words used in the listings and figures of all the FORTH articles in this issue. It does not include all the standard words in FORTH (there are quite a few), nor does it include user-defined words required by each article. The **pronunciations of some words are given in parentheses.** Wherever possible, an example is given showing the use of the defined word. The words "before" and "after" show the stack before and after the word is executed. In these representations of the stack, the top of the stack is the rightmost number, and the words influenced by the defined word are depicted in boldface.

The columns marked "uses" and "leaves" show how the execution of a FORTH word affects the top entries of the stack. FORTH words remove the stack entries they use and sometimes leave one or more entries on the stack. Therefore, the number under "uses" and "leaves" should equal the number of entries in boldface in the "before" and "after" stacks. Asterisks in both columns mean that the numbers are not given for multiword constructs for the purpose of clarity.

Multiword constructs, like the following example:

{ IF ... ELSE ... THEN }

are enclosed in braces with the keywords separated by ellipses that represent zero or more FORTH words. Also, these constructs are listed only under the first word of the construct. In general, all the words in this table are sorted by ascending ASCII value — for example, the word * (ASCII hexadecimal 2A) is listed before the word + (ASCII hexadecimal 2B).

This glossary assumes that the output device used by the FORTH system is a video terminal. When any definition refers to the video display or display, it actually refers to whatever output device or devices are currently enabled.

FORTH Glossary

Word	Uses	Leaves	Notes
{ ! } (store)	2	0	Sees top-of-stack as address of a 2-byte variable and stores second-on-stack in this variable; for example, suppose that address 20000 points to a 2-byte variable; then: before: 9 9 **−1150 20000** after: 9 9 (−1150 is stored in a 1-byte variable.)
{ " }	0	0	{ " HI THERE!" }, when executed, prints HI THERE! on the video display.
{ ' } (tic)	0	1	Puts onto top-of-stack the address of the word that *follows* it.
{ (}	0	0	{ (THIS IS A COMMENT) } , if included in a definition, will not be compiled; { (} requires a {) } to end the comment.
*	2	1	Multiplication; example: before: 9 9 **3 5** after: 9 9 **15** The word * multiplies 5 and 3, leaving 15.
+	2	1	Addition; example: before: 9 9 **3 5** after: 9 9 **8** The word + adds 5 and 3, leaving 8.
{ , }	1	0	Embeds the number on the top of the stack into a dictionary definition, incrementing the dictionary pointer.
−	2	1	Subtraction; example: before: 9 9 **3 5** after: 9 9 **−2** The word − subtracts 5 from 3, leaving −2.
{ . }	1	0	Displays the number on the top of the stack; example: before: 9 9 3 5 after: 9 9 3 (5 is printed on screen.)

YOU KNOW WATSON, HAYDEN SOFTWARE OFFERS THE BEST REASONS TO OWN A MICROCOMPUTER...

I had heard everyone speak about Hayden software. How it was the finest available. "I would be so pleased," said I, "if we could discuss some of their programs."

"Sounds delightful, Watson. This **SARGON II** plays a marvelous game of chess. It has 7 levels of play, and levels 0-3 play in tournament time. It has a randomized opening book for all 7 levels of play through 3 moves. And, a special hint mode is included at all levels of play but 0. Imagine that, old fellow. Small wonder they call it, 'the champ of champs.'"
(#03403, TRS-80 Level II; #03404, Apple II; each $29.95; #03409, Apple II Disk Version; #03408, TRS-80 Disk Version; each $34.95)

I then looked around, and spotted a new program. I lifted the cassette, examined it critically, and then began to speak. "This **DATA MANAGER** looks to be as fine a specimen as that **SARGON II** . It stores up to 96,000 alphanumeric characters on just one floppy disk. And one third of this information may be recovered from Random Access Memory at a time. This means, that on just eleven diskettes one can store and retrieve up to 1,000,000 characters. It is, in my judgment, a clever program to have around."
(#04909, Apple II Disk, $49.95)

"Extraordinary. Here's another program for the Apple II. They call it **APPLESOFT UTILITY PROGRAMS.** It contains 9 subroutines, among them 3 statement formatters: REM, PRINT, and POKE writers. You can calculate the decimal address of your machine language programs, get an exact byte and line count, renumber the program in any increment, and much more. I wonder what other fine programs are to be had from Hayden?" **(#03504, Apple II, $29.95)**

Holmes leaned back, still puffing at his black pipe. "Wait a minute," said he. "Here's something."

"What is it?" said I.

"It's **APPLE™ ASSEMBLY LANGUAGE DEVELOPMENT SYSTEM.** It features a cursor-based editor, global and local labels, and disk-based macros which allow you to incorporate subroutines into any program. And, one can write and modify machine language programs quickly and easily. It is indeed quite remarkable."
(#04609, Apple II Disk Version, $39.95)

"Quite. But let us not forget **BLACKJACK MASTER,** what. Unlike other blackjack programs that emphasize graphics and harmless fun, this is a serious game. Imagine being able to perform complex simulations and evaluations of any playing and betting strategies that are entered into the microcomputer. And, it will tutor one in how to play these strategies! Good gracious, there's also a **$250.00 BLACKJACK Challenge!**
(#05303, TRS-80 Level II, $19.95; #05308, TRS-80 Disk Version, $24.95)

"What is that ?"

"My dear fellow, just see the package for details! Holmes, you do agree then, that these programs are a fine lot?"

"I'm satisfied, Watson."

"A very sensible reply, Holmes. It's simple to see why one would want to own a microcomputer when Hayden software is available. It's easy-to-use, ready-to-run, comes with full documentation, and can be had at any local computer store."

"Or Watson, you can call **TOLL FREE,** 24 hours a day, (1-800-827-3777, ext. 302)* **TO CHARGE YOUR ORDER TO** Master Charge or Visa! Minimum order is $10.00; customer pays postage and handling."

"Splendid, Holmes, simply splendid."

Hayden Book Company, Inc.

50 Essex Street, Rochelle Park, NJ 07662

*From Missouri, call (1-800-892-7655, ext. 302)

Apple is a trademark of Apple Computer Company, Inc. and is not affiliated with Hayden Book Company, Inc.

Circle 130 on inquiry card.

/	2	1	Division; example: before: 9 9 **13** **2** after: 9 9 **6** The word / divides 13 by 2, leaving 6. (Remainder is lost.)
0<	1	1	If top-of-stack is <0, it is replaced with a 1 (true); if top-of-stack is ≥ 0, it is replaced with a 0 (false); example: before: 9 9 3 **5** after: 9 9 3 **0**
1+	1	1	Adds 1 to top-of-stack; example: before: 9 9 3 **5** after: 9 9 3 **6**
{ : ... ; }	*	*	{ : } begins the definition of a word; { ; } ends the definition; example: { : 3* 3 * ; } defines the word 3*.
=	2	1	If the two top items on the stack are exactly equal, both of them are removed and replaced with a single 1 (true); if not, both are replaced with a single 0 (false); example: before: 9 9 **3** **5** after: 9 9 **0**
<	2	1	If the second item on the stack is less than the top item on the stack, both of them are removed and replaced with a single 1 (true); if not, both of them are replaced with a single 0 (false); example: before: 9 9 **3** **5** after: 9 9 **1**

StackWork's

FORTH

A full, extended FORTH interpreter/compiler produces COMPACT, ROMABLE code. As fast as compiled FORTRAN, as easy to use as interactive BASIC.

SELF COMPILING
Includes every line of source code necessary to recompile itself.

EXTENSIBLE
Add functions at will.

CP/M* COMPATIBLE

Z80 & 8080 ASSEMBLERS included

Single license
Supplied with extensive user manual and tutorial:
$150.00
Documentation alone: $25.00

OEM's, we have a deal for you!

CP/M Formats: 8" soft sectored, 5"
Northstar, 5" Micropolis Mod II.

Please specify CPU type
Z80 or 8080

All Orders and General Information:
SUPERSOFT ASSOCIATES
P.O. BOX 1628
CHAMPAIGN, IL 61820
(217) 359-2112
Technical Hot Line: (217) 359-2691
(answered only when technician is available)

VISA

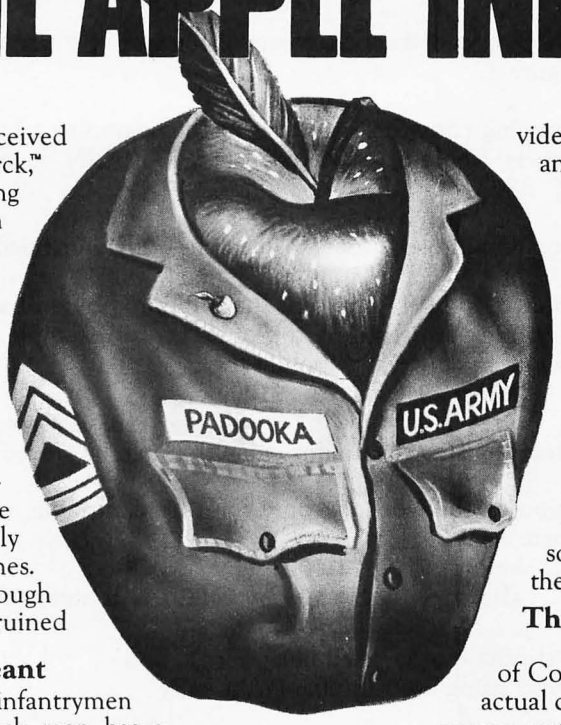
master charge
THE INTERBANK CARD

SuperSoft
First in Software Technology

*CP/M registered trademark Digital Research

{ <BUILDS ... DOES> }	*	*	Used to define new defining words; see "FORTH Extensibility" article, figure 4.
>	2	1	Similar to entry for < ; example: before: 9 9 **3** **5** after: 9 9 **0** (3 is not less than 5.)
{ ? }	1	0	Sees top-of-stack as address for 2-byte variable; displays value of that variable; using the example for { ! } , then: before: 9 9 **20000** after: 9 9 (−1150, contents of 20000, prints on screen.)
@ (fetch)	1	1	Sees top-of-stack as address for 2-byte variable and replaces it with value of that variable; using the example in { ! } : before: 9 9 **20000** after: 9 9 **−1150** (−1150 is contents of 2-byte variable at 20000.)
ALLOT	1	0	Sees top-of-stack as number of bytes to be reserved (and filled in later) *during the definition of a word.*
AND	2	1	Does an AND operation on the corresponding bits of the top two stack entries (both 16-bit numbers); example: before: 9 9 **3** **5** after: 9 9 **1** (3 AND 5, in binary, is 1.)
BASE	0	1	BASE is a *1-byte variable* that contains the number base being used; for example, { 2 BASE C! } causes all subsequent input and output to be in binary (base 2); execution of this word causes the address of this 1-byte variable to be placed on top-of-stack.

JOIN THE APPLE INFANTRY!



Judging by the letters we've received from buyers of Computer Bismarck,™ home computer historical wargaming is a great mind-stretching recreation to uncramp the old synapses after a few hours of trying to cram 54K of code into 48K of memory. But before you read any further, let us warn you that our *new* game, Computer Ambush,™ is more gut-wrenching than mind-stretching.

Strategy versus Tactics

Computer Bismarck is a "strategic" wargame, casting you in the role of a British or German admiral coolly deploying fleets of ships and planes. Computer Ambush is "tactical"...tough and dirty street fighting in a half-ruined French town.

You're a Sergeant

You command a squad of ten infantrymen (either American or German). Each man has a name, rank, and such individual combat skills as footspeed, strength, intelligence, endurance and marksmanship...all of which affect the success of every move you order. Your squad is armed with grenades, rifles, automatic weapons, plastic explosives, bayonets, and even garottes. You fight with carefully-aimed shots, area bursts, explosions, and hand-to-hand combat. They can result in wounds or deaths, depending on time, distance, the individual skills of each soldier, and your ability as a squad leader.

Battlefield

Street fighting is the most challenging tactical command situation in modern warfare. Using "Higher Text", a character generator, the computer displays a map showing buildings (your plastic explosives can turn them into rubble during the game), walls, hedges, doors, windows (nasty sniper positions), and each of your men by name. The enemy is usually hidden.

Play the Computer

The computer plays the German squad leader (*Feldwebel* Kurt Reich) to perfection. It defends the town with sniping, machine guns, grenades, and finally, with hand-to-hand combat.

You're Sergeant Buck Padooka. You maneuver your men and fire at revealed and probable German positions. If you kill all the Germans before they get you, the town is yours. But the computer's a tough, experienced squad leader, so don't expect to win very often.

Play a Friend

You take turns examining the video map display, moving your men, and firing weapons. Your options are limited by casualties, wounds, physical exhaustion, ammo supplies, terrain, and the individual skills of each of your men. The same is true for your opponent. And every action takes precious time, even the flight of a grenade or bullet. (Remember, time is life or death on the battlefield *and* in Computer Ambush!) After each turn, the computer displays the movements and weapons fire of both squads as tracks on the video map...just once, so watch carefully to figure out where the enemy is, or was.

The Sweat and Death of War

The time pressure and complexity of Computer Ambush create the stress of actual combat command. Your palms sweat as you watch PFC Chuck Lawson get blown away by that damned Kraut machine gun you forgot when you ordered him to sneak across the alley. If you can imagine a game that's more complex than chess, requires much faster decision-making, rewards courage and cruelly punishes foolhardiness...that's Computer Ambush!

$59.95 and an Apple

If you've got an Apple II Plus (or an Apple III or an Apple II with Applesoft Firmware ROM Card) with 48K memory and a 5¼ inch mini-floppy disc drive, you can be playing Computer Ambush in a few days. For $59.95, you get the game program disc; 2 mapboard charts (for plotting strategies in grease pencil while your opponent is at the computer); 2 squad leader's data cards; and a rule book. You also get a game selection card which tells you how to set up any of seven wargames: NCO Training, Ambush or Raid against the computer; and Patrol, Ambush, Strongpoint, or Free Form against a human opponent.

Call **800-648-5600 (toll free)**, and ask Operator 181 to charge Computer Ambush (or Computer Bismarck) to your VISA or MASTERCHARGE. In Nevada call 800-992-5710. To order by mail, send your check to Strategic Simulations Inc., Dept. B, 450 San Antonio Road, Suite 62, Palo Alto, CA 94306.

With our 14-day money back guarantee, your satisfaction is assured. So come and join our Apple Infantry!

SSI

COMPUTER AMBUSH™...You've got a war on your hands.

Circle 136 on inquiry card.

{ BEGIN ... UNTIL }	*	*	Looping construct that tests at the end of the loop; see "What Is FORTH?" article, figure 4.
{ BEGIN ... WHILE ... REPEAT }	*	*	Looping construct that tests at the beginning of the loop; see "What is FORTH?" article, figure 5; other forms are { BEGIN ... PERFORM ... PEND } and { BEGIN ... IF ... WHILE }.
{ C; }	*	*	Sometimes used to end a machine-code word definition; most versions use NEXT.
{ C! }	2	0	Similar to { ! } except that only low byte of second-to-top is stored in 1-byte variable pointed to by top-of-stack; for example, suppose that address 21000 points to a 1-byte variable; then: before: 9 9 **103** **21000** after: 9 9 (103 is stored in 1-byte variable.) Note that the maximum value that can be stored in 1 byte is 127.
C@	1	1	Same as the word @, only for 1-byte variable; using the example of { C! }, then: before: 9 9 **21000** after: 9 9 **103** (103 is contents of 1-byte variable at 21000.)
{ CODE ... NEXT }	*	*	Defining words, used like { : } and { ; }, used when defining a new word using assembly language only.
CONSTANT	1	0	Creates a constant that has the value of top-of-stack; for example, before executing the phrase { CONSTANT CON }, the stack looks like: 9 9 **25140**

Put your applications to work on the Mostek STD-Z80 BUS.

You're ready. You've gone beyond the learning stage and are using your personal computer to implement real time control applications.

We can take you one big step further. By showing you how to take the programs you have developed using your TRS-80 (or other Z80 based computer) and place them in PROM on a low-cost stand-alone micro card system. This will not only free up your main computer for new applications, but will also permit your current application to be "on-line" continuously, or even "cloned"-for multiple installations or sales to other users.

Mostek's MD Series™ of STD-Z80 BUS compatible microcomputer cards makes all this possible. There are more than twenty different boards in this off-the-shelf family available now, including data processing boards; memory boards (Static and Dynamic RAM, ROM/PROM); I/0 cards; A/D; D/A; high speed floating point math; and floppy disk controller cards.

QC Micro Systems offers all of these products directly, including a full range of support products such as prototyping hardware, support software and, of course, extensive documentation. And much more. Contact us for details so they can be put to work for you, by sending in the coupon below. Today. QC MicroSystems, P. O. Box 401326, Garland, TX 75040, (214) 343-1282.

QC Micro Systems

Yes! I'm ready for more information on the STD-Z80 BUS products from QC MicroSystems. My application is

☐ Personal ☐ Resale
☐ Industrial Control ☐ Other_____
☐ Yes, I would like to use my TRS-80 as my STD BUS development station.

Name_____

Company_____

Address_____

City_____ State_____ Zip_____

Circle 139 on inquiry card.

			After the phrase has been executed, the stack looks like: 9 9 and the word CON , when executed, will place 25140 on the top of the stack.
CR	0	0	Causes the cursor to jump to the beginning of the next line of the display.
{ DO ... LOOP }	2	0	Looping construct that specifies a beginning and an ending-value-plus-one; see "What Is FORTH?" article, figure 3.
DROP	1	0	Drops top entry from stack; example: before: 9 9 3 5 after: 9 9 3
DUP	1	2	Duplicates item on top-of-stack; example: before: 9 9 3 5 after: 9 9 3 5 5
ECHO	1	0	Isolates the low-order byte of the 2-byte entry on top of the stack and writes it to the video display; example: before: 9 9 **32** after: 9 9 (A space, ASCII decimal 32, is printed.) ECHO is named EMIT in some versions.
FILL	3	0	Fills an area of memory with a given value; for example, { 255 3000 100 FILL } fills memory locations from 3000 thru 3099 (100 bytes) with the value 255.
FORGET	0	0	Causes system to delete all definitions including and after the word following FORGET ; for example, { FORGET BASEPGM } causes the system to delete BASEPGM and all FORTH words, variables, and constants defined after it.

THE ORIGINAL MAGAZINE FOR OWNERS OF THE TRS-80™* MICROCOMPUTER

SOFTWARE FOR TRS-80™ OWNERS

H & E COMPUTRONICS INC.

MONTHLY NEWSMAGAZINE FOR TRS-80™ OWNERS

MONTHLY NEWSMAGAZINE
Practical Support For Model I & II

- PRACTICAL APPLICATIONS
- BUSINESS
- GAMBLING • GAMES
- EDUCATION
- PERSONAL FINANCE
- BEGINNER'S CORNER
- NEW PRODUCTS
- SOFTWARE EXCHANGE
- MARKET PLACE
- QUESTIONS AND ANSWERS
- PROGRAM PRINTOUTS

. . . . AND MORE

PROGRAMS AND ARTICLES PUBLISHED IN OUR FIRST 12 ISSUES INCLUDE THE FOLLOWING:

- A COMPLETE INCOME TAX PROGRAM (LONG AND SHORT FORM)
- INVENTORY CONTROL
- STOCK MARKET ANALYSIS
- WORD PROCESSING PROGRAM (FOR DISK OR CASSETTE)
- LOWER CASE MODIFICATION FOR YOUR VIDEO MONITOR OR PRINTER
- PAYROLL (FEDERAL TAX WITHHOLDING PROGRAM)
- EXTEND 16-DIGIT ACCURACY TO TRS-80™ FUNCTIONS (SUCH AS SQUARE ROOTS AND TRIGONOMETRIC FUNCTIONS)
- NEW DISK DRIVES FOR YOUR TRS-80™
- PRINTER OPTIONS AVAILABLE FOR YOUR TRS-80™
- A HORSE SELECTION SYSTEM***ARITHMETIC TEACHER
- COMPLETE MAILING LIST PROGRAMS (BOTH FOR DISK OR CASSETTE SEQUENTIAL AND RANDOM ACCESS)
- RANDOM SAMPLING***BAR GRAPH
- CHECKBOOK MAINTENANCE PROGRAM
- LEVEL II UPDATES***LEVEL II INDEX
- CREDIT CARD INFORMATION STORAGE FILE
- BEGINNER'S GUIDE TO MACHINE LANGUAGE AND ASSEMBLY LANGUAGE
- LINE RENUMBERING
- AND CASSETTE TIPS, PROGRAM HINTS, LATEST PRODUCTS COMING SOON (GENERAL LEDGER, ACCOUNTS PAYABLE AND RECEIVABLE, FORTRAN-80, FINANCIAL APPLICATIONS PACKAGE, PROGRAMS FOR HOMEOWNERS, MERGE TWO PROGRAMS, STATISTICAL AND MATHEMATICAL PROGRAMS (BOTH ELEMENTARY AND ADVANCED) . . . AND

FREE

WORD PROCESSING PROGRAM **(Cassette or Disk)** For writing letters, text, mailing lists, etc., with each new subscriptions or renewal.

LEVEL II RAM TEST **(Cassette or Disk)** Checks random access memory to ensure that all memory locations are working properly.

DATA MANAGEMENT SYSTEM **(Cassette or Disk)** Complete file management for your TRS-80™

CLEANUP **(Cassette or Disk)** Fast action Maze Game

ADVENTURE **(Cassette or Disk)** Adventure #0 by Scott Adams (From Adventureland International)

FREE

* TRS-80™ IS A TRADEMARK OF TANDY CORP.

SEND FOR OUR NEW 48 PAGE SOFTWARE CATALOG (INCLUDING LISTINGS OF HUNDREDS OF TRS-80™ PROGRAMS AVAILABLE ON CASSETTE AND DISKETTE). $2.00 OR **FREE** WITH EACH SUBSCRIPTIONS OR SAMPLE ISSUE.

H & E COMPUTRONICS INC.
MATHEMATICAL APPLICATIONS SERVICE™

50 N. PASCACK ROAD
SPRING VALLEY, NEW YORK 10977

master charge

24 HOUR ORDER LINE
(914) 425-1535

VISA

NEW TOLL-FREE ORDER LINE
(OUTSIDE OF N.Y. STATE)
(800) 431-2818

ONE YEAR SUBSCRIPTION $24 .....
TWO YEAR SUBSCRIPTION $48 .....
SAMPLE OF LATEST ISSUE $ 4 .....
START MY SUBSCRIPTION WITH ISSUE .....
(#1 - July 1978 • #7 - January 1979 • #12 - June 1979 • #18 - January 1980)
NEW SUBSCRIPTION ..... RENEWAL .....

NEW!!!
MOD-II NEWSLETTER
$12/year (or 12 issues)

CREDIT CARD NUMBER _____ EXP. DATE _____

SIGNATURE _____

NAME _____

ADDRESS _____ CITY _____ STATE _____ ZIP _____

*** ADD $6/YEAR (CANADA, MEXICO) - ADD $12/YEAR AIR MAIL - OUTSIDE OF U.S.A., CANADA & MEXICO ***

Circle 142 on inquiry card.

H	0	1	*2-byte variable* containing address of the top of the dictionary; execution of this word causes the *address* of the variable H (not its value, which equals the address of the top of the dictionary) to be placed on top of the stack.
HERE	0	1	Places the address of the next byte to be used in the dictionary (the *value* of H) on top of the stack.
I	0	1	When executed within a { DO ... LOOP } , the word I pushes onto the top of the stack the value of the index counter; for example, { 10 0 DO I . LOOP } prints the numbers from 0 thru 9.
{ IF ... ELSE ... THEN }	1	0	Conditional execution of words depending on value of top-of-stack. If nonzero, execute words between IF and ELSE . If zero, execute words between ELSE and THEN ; for example, { IF " NUMBER ON TOP IS NONZERO" ELSE " NUMBER ON TOP IS ZERO" THEN } prints the appropriate message depending on the value on top of the stack.
KEY	0	1	Gets a single character from the keyboard; for example, if the stack before we press the space bar is: 9 9 3 5 Then, after we press the space bar (ASCII value decimal 32), the stack is: 9 9 3 5 **32**
MAX	2	1	Compares the two top entries on the stack and leaves only the larger; example: before: 9 9 **3 5** after: 9 9 **5**
MIN	2	1	Compares the two top entries on the stack and leaves only the smaller; example: before: 9 9 **3 5** after: 9 9 **3**
MINUS	1	1	Changes the sign of the entry on top of the stack; example: before: 9 9 3 **5** after: 9 9 3 **−5**
OVER	2	3	Copies the second-to-top entry onto the top of the stack; example: before: 9 9 **3 5** after: 9 9 **3 5 3**
PAD	0	1	PAD is a *2-byte variable* that points to the beginning of a 64-byte area for temporary storage of character strings; execution of this word causes the *address* of this 2-byte variable to be placed on top of the stack.
SWAP	2	2	Exchanges the two top entries on the stack; example: before: 9 9 **3 5** after: 9 9 **5 3**
U*	2	1	The *lower 8 bits* of the two top entries on the stack are isolated and multiplied together, leaving their unsigned 16-bit product; example: before: 9 9 **3 5** after: 9 9 **15** Each factor will effectively be 255 or less, giving a product that will not overflow in 16 bits.
VARIABLE	1	0	Creates a variable that has the value of top-of-stack; example, before executing the phrase { VARIABLE VAR } , the stack looks like: 9 9 −14017 After the phrase has been executed, the stack looks like: 9 9 and the word VAR , when executed, will place the *address* of the variable on the stack. (The 2-byte number stored at that address will contain the value −14017.) Unlike a constant, the value of a variable can be changed using { ! } (store).
{] }	*	*	Resumes compilation of a colon definition. ■