

BYTE[®]

the small systems journal
A MCGRAW-HILL PUBLICATION

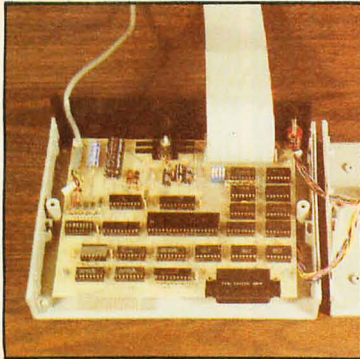
: DOUBLE

+

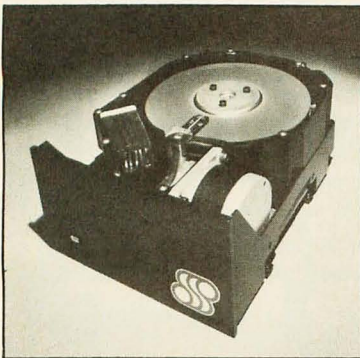
DUPLIC

ROBERT
30 TIMNEY

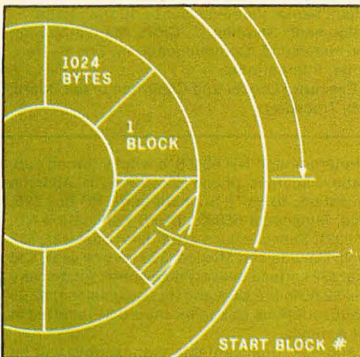
THE FORTH LANGUAGE



Page 22



Page 58



Page 164



Page 210

Foreground

22 A BUILD-IT-YOURSELF MODEM FOR UNDER \$50

by *Steve Ciarcia*

This originate-only modem will allow you to get started in intercomputer communication with minimal expense.

58 THE HARD-DISK EXPLOSION: HIGH-POWERED MASS STORAGE FOR YOUR PERSONAL COMPUTER

by *Tom Manuel*

Thanks to new hard-disk technology, personal computer users can add millions of bytes of mass storage to their systems at a reasonable cost.

100 WHAT IS FORTH? A TUTORIAL INTRODUCTION

by *John S James*

Here is an overview of FORTH that lays the foundation for the other theme articles in this BYTE.

150 BREAKFORTH INTO FORTH

by *A Richard Miller and Jill Miller*

If you can't imagine any personal use for FORTH, can you imagine a 96-line program that plays a fast, animated game with sound on the TRS-80?

164 FORTH EXTENSIBILITY: OR HOW TO WRITE A COMPILER IN TWENTY-FIVE WORDS OR LESS

by *Kim Harris*

This tutorial explains the capability for defining new families of FORTH words.

210 CONSTRUCTION OF A FOURTH-GENERATION VIDEO TERMINAL, PART 1

by *Theron Wierenga*

Part 1 of this article presents a new design using the 8275 controller and a dedicated Z80 microprocessor.

Background

76 THE EVOLUTION OF FORTH, AN UNUSUAL LANGUAGE

by *Charles H Moore*

The inventor of the language recalls its design and how it evolved over a 10-year period.

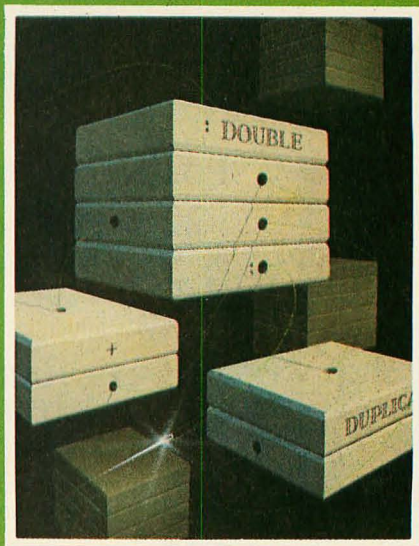
198 KHACHIYAN'S ALGORITHM, PART 1: A NEW SOLUTION TO LINEAR PROGRAMMING PROBLEMS

by *G C Berresford, A M Rockett, and J C Stevenson*

Now you can study the algorithm that promised to revolutionize linear programming.

Nucleus

- 6 Editorial: Threads of a FORTH Tapestry
- 14 Letters
- 40 Product Review: The Ohio Scientific CA-15 Universal Telephone Interface
- 46 Product Review: The Heath H-89 Computer
- 72 Programming Quickies: Self-Reproducing Programs
- 94 BYTELINES
- 98 Selected FORTH Vendors
- 196 A FORTH Glossary
- 226 Clubs and Newsletters
- 230 Event Queue
- 234 Ask BYTE
- 248 What's New?
- 302 Unclassified Ads
- 303 BOMB, BOMB Results
- 304 Reader Service



ON THE COVER

This month's cover by Robert Tinney shows a rocket-like needle threading its way through granite cubes labeled: DOUBLE , DUPLICATE , and + . The threaded path of the needle is a representation of the process used in FORTH and other threaded languages to create a new word (here, DOUBLE) with previously defined words (here, DUPLICATE and +).

Other aspects of this fascinating language are described in the editorial, "Threads of a FORTH Tapestry," and in the theme articles for this issue.

Editorial

Threads of a FORTH Tapestry

Editor's Note: *This month's editorial is by BYTE Editor Gregg Williams. Gregg was responsible for the preparation of this month's special section devoted to the FORTH language. Carl Helmers returns next month with an editorial....CM*

What do a portable heart monitor, the new Craig Language Translator, a peach-sorting machine, and a movie called *Battle Beyond the Stars* have in common? The answer is FORTH, a not-so-new language as comfortable in industrial machinery as it is in a personal computer. In fact, it was originally used by its inventor, Charles H Moore, to control the telescope and equipment at the Kitt Peak Observatory.

Although I have known about FORTH for about a year, it was only during the preparation of this issue that I began to actively keep my ears open for mention of this unusual language. I have uncovered a lot of information (and some experience) about FORTH and its variations. The language is so unusual that no single line of thought could give you a picture of what the language is like. Instead, the following sections represent several threads from the rich tapestry called FORTH.

FORTH in the Real World

No language I know of is as comfortable in real-world situations as FORTH. Here are some examples of the breadth of applications that have been created using FORTH:

- Elicon Inc of Brea, California, is using FORTH software to drive the same kind of computer-controlled cameras that were used to film the sophisticated space-battle scenes in *Star Wars*. New World Productions of Venice, California, is using this camera system to film the spaceship sequences in the motion picture *Battle Beyond the Stars*. In a related development, Magicam Inc (which devised a number of the special effects for the recent movie *Star Trek*) is in the process of converting control of its master-slave camera pair from an analog computer to a digital computer running FORTH software. In the Magicam process, the master camera follows actors on a special blue stage while the computer guides the slave camera across a detailed model. Later, the two images are optically combined, producing the effect of the actors actually being in the landscape depicted on the model.

- Allen Test Products of Kalamazoo, Michigan, has developed an ignition analyzer for use in service stations and automobile repair shops that analyzes the behavior of automobile ignition systems and displays both diagnostic and corrective information. Formerly, the voltage waveform from a spark plug was displayed on an oscilloscope, after which a mechanic would attempt repairs based on his interpretation of the waveforms.

- Atari Inc is using FORTH in two of its divisions and is rumored to be contemplating other uses for the language. In its Coin-Operated Division,

which develops and markets the stand-alone games found in pinball arcades and restaurants, a 6502-based development system employs FORTH software to debug and test arcade circuit boards. In addition, Atari has developed its own custom version of the language, called game-FORTH, that is awaiting its first use to replace machine code as the language used to create arcade games. Someday soon, you may play a coin-operated game without knowing that you are actually running a FORTH program.

In the Consumer Group of Atari, a version of FORTH that has been extended to allow manipulation of the video screen and game peripherals has been developed for the Atari 800 computer. Although no definite plans have been made, Atari may market it as an option for the Atari 800, or, like the Coin-Operated Division, use it in a "transparent" mode to implement games and other programs.

● FORTH is used in a portable 1802-based computer that aids in the treatment of patients with infrequent heart flutter. The device, small enough to be worn comfortably by

the patient during his or her daily activities, constantly updates a "snapshot" of the patient's heart activity every 7 seconds. In addition to recording this information in real time, the device analyzes the data for evidence of a heart murmur. When a murmur is detected, the device stores the data containing the evidence and signals the patient to return with the device to the doctor's office for analysis and diagnosis.

● In another medical application, FORTH is the sole language used in a computer at the Cedar-Sinai Medical Center in Los Angeles, California. Using FORTH, a Digital Equipment Corporation PDP-11/60 simultaneously performs, among others, the following tasks: manages 32 remote terminals; stores patient information from an optical reader into a large data base; runs a statistical package that analyzes the patient data base in search of trends in the physical makeup, treatment, and results of similar patients; and analyzes blood samples and heart behavior in real time while a patient is exercising on a treadmill machine. Spencer SooHoo, in the pulmonary

medicine section, is also developing a portable 6800-based FORTH system to be used for monitoring intensive-care patients.

● A stripped-down version of FORTH was used to create the hand-held Craig M100 Language Translator under time, size, and other design constraints. This same language also runs the software inside the translator unit. In a related product, a hand-held ASCII terminal manufactured by MSI Data Corporation of Costa Mesa, California, also uses FORTH internally.

● In what must be the most interesting FORTH application I have encountered, a central California fruit farming cooperative uses an 8080-based machine running FORTH to adaptively sort and grade peaches. Infrared sensors send information to the computer on the coloring and quality of pitted peach halves that pass the sensors on a conveyer belt. After analyzing this data, the FORTH program causes flippers to knock the peach halves into appropriately graded bins—extra fancy, fancy, etc. In addition, the program keeps track of the percentage of peaches in each bin and changes its selection criteria to maintain a certain fixed ratio among the various grades of peaches.

● Last but not least, FORTH is used in several aerospace applications. A FORTH-like language called IPS (running on an 1802-based system) is orbiting Earth in an amateur radio satellite called the OSCAR Phase III. Avco Inc is using another 1802-based system (again, for the small size and power consumption of the 1802 microprocessor) to monitor temperature and take care of ground-to-satellite and satellite-to-ground telemetry in a military satellite.

Who Should Try FORTH?

FORTH is an easy language: a high school student, Arnold Schaeffer, wrote an arcade-type game called BREAKFORTH. (See "Breakforth into FORTH," by A Richard Miller and Judy Miller, on page 150.)

FORTH is a difficult language: it easily beats APL as a "write-only language"; you can write a program in the language, but you can't easily read what you've written.

Given these two valid extremes, your initial reaction might be, "This doesn't make sense." True, learning

A CREATION OF COMPUTER HEADWARE

WHATSIT?™

(Wow! How'd All That Stuff get In There?)

A sophisticated, self-indexing filing system—flexible, infinitely useful and easy to use, that adapts to your needs.

WHATSIT comes ready to run on your Apple, NorthStar, or CP/M computer. See your dealer...or write or call:

HARDHAT
Software

P.O. Box 14815 • San Francisco, CA 94114 • Tel: (415) 621-2106

FORTH takes some time; it's somewhat like learning a foreign language. So far, my experiences with FORTH remind me of my attempts at learning a smattering of Russian; both languages are so different from any I've seen before—French or Spanish, BASIC or FORTRAN—that I have to mentally shift gears to work in the new language.

You should give FORTH a try if you are excited by what you see here. Especially important in this respect are the articles, "What is FORTH? A

Tutorial Introduction," by John James, and "A FORTH Glossary," pages 100 and 186, respectively. Your best bet is to get to a computer that can run a version of FORTH; or, better yet, get someone who knows the language to demonstrate it to you.

My first experience with FORTH was at the Fourth West Coast Computer Faire in May 1979. A member of the FORTH Interest Group was demonstrating the language using an Apple II and an Advent television screen. First, he defined a word called

COUNT, like this:

```
: COUNT 0 DO I . LOOP ;
```

Then he said { 6 COUNT } (note: the braces are not part of the expression; see the accompanying text box), the computer replied with { 0 1 2 3 4 5 OK }. I was instantly hooked on learning more about FORTH. What he had done closely paralleled the *iota* function in APL, and anything that even resembled APL was going to get my full attention.

If you are at all dissatisfied with the capabilities of your current computer, or if you feel that there should be more to computers than BASIC and assembly language, you should try FORTH. Once you get accustomed to its peculiar syntax, you can make it do nearly anything you want it to. In fact, you can even make it have features it did not previously have. Assembly language is like this to some extent, but FORTH is a higher-level language with the same abilities—only magnified. FORTH is what I call a "homebrew" language; its enthusiasts carry with themselves the same look-how-this-works enthusiasm as do most hardware hackers who build their own hardware. If we ever have a homebrew software issue, FORTH will certainly be included.

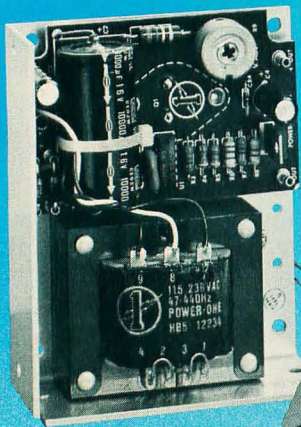
FORTH is the ultimate software hacker's language because, like a bag of components before a hardware hacker, you can do anything you want to with it. It can be argued that assembly language is the ultimate programming language; strictly speaking, this is true, but it takes so much more time to craft a piece of software in assembly language that it is practically ruled out in most cases.

However, this total freedom carries with it complete responsibility. Since, for example, the FORTH program you write is free to use an array subscript that is out of bounds, you must be responsible enough to either (a) put in error-checking routines (you can take them out later), or (b) build your program up from small tested modules to assure that your program will never execute an improper subscript. If you would rather have the language system do this kind of work for you, stick to BASIC or whatever you're running now.

Text continued on page 128

\$24.95

...AND HOLDING



Model HB5-3/OVP

5V at 3A with Built-in OVP

Power One's B Case models started at \$24.95. Over 100,000 models and five years later, they're still only \$24.95!

- 115/230 VAC Input
- OVP Built-in
- .05% Regulation
- 2-Year Warranty
- 2-Hour Burn-in
- UL Recognized
- CSA Certified



Get all the details on our 84 standard open frames in our new 1978 catalog.

IN-STOCK NATIONWIDE... FOR IMMEDIATE DELIVERY

EASTERN REGIONAL SALES OFFICE: Schenectady, N.Y. (518) 399-9200 **ALA.:** Huntsville, Rakes Engr. & Marketing Corp. (205) 883-9260 **ARIZ.:** Phoenix, PLS Assoc. (602) 279-1531 **CAL.:** Pasadena, A-F Sls. Engr. (213) 681-5631; San Diego, A-F Sls. Engr. (714) 226-8424; San Jose, Richards Assoc. (408) 246-5860 **COL.:** Denver, PLS Assoc. (303) 773-1218 **CT.:** Litchfield, Digital Sls. Assoc. (203) 567-9776 **FLA.:** Orlando, OEM Marketing Corp. (305) 299-1000 **GA.:** Duluth, Rakes Engr. & Marketing Corp. (404) 476-1730 **ILL.:** Chicago, Coombs Assoc. (312) 298-4830 **IND.:** Indianapolis, Coombs Assoc. (317) 897-5424 **MD.:** Wheaton, Brimberg Sls. Assoc. (301) 946-2670; Baltimore, Brimberg Sls. Assoc. (301) 792-8661 **MASS.:** Waltham, Digital Sls. Assoc. (617) 899-4300 **MICH.:** Southfield, L.H. Dickelma Co. (313) 353-8210 **MINN.:** Minneapolis, Engr. Prod. Assoc. (612) 925-1883 **N.J.:** Whippany, Livera-Polk Assoc. (201) 377-3220; Marmora, Holdsworth (609) 398-4340 **N.M.:** Albuquerque, PLS Assoc. (505) 255-2330 **N.Y.:** Roslyn Hts., Livera-Polk Assoc. (516) 484-1276; Syracuse, C.W. Beach (315) 446-9587 **N.C.:** Charlotte, Over & Over Inc. (704) 527-3070 **OHIO:** Cleveland, Marlow Assoc. (216) 991-6500; Dayton, Marlow Assoc. (513) 434-5673 **OKLA.:** Tulsa, Advance Technical Sls. (918) 743-8517 **ORE.:** Portland, Jas. J. Backer (503) 297-3776; Salem, Jas. J. Backer (503) 362-0717 **PENN.:** Pittsburgh, Marlow Assoc. (412) 831-6113; Newtown Sq., Holdsworth & Co. (215) 356-8550 **TEX.:** Dallas, Advance Technical Sls. (214) 361-8584; Solid State Electr. (214) 352-2601; Houston, Advance Technical Sls. (713) 469-6668; Solid State Electr. (713) 772-8483 **UTAH:** Salt Lake City, PLS Assoc. (801) 466-8729 **WASH.:** Seattle, Jas. J. Backer (206) 285-1300; Radar Elec. Co. (206) 282-2511 **WIS.:** Milwaukee, Coombs Assoc. (414) 671-1945 **EUROPE:** Hanex, L.A., CA (213) 556-3807 **CANADA:** Duncan Instr., Weston, Ontario (416) 742-4448; Winnipeg, Manitoba, Cam Gard Supply Ltd. (204) 786-8481



Power One Drive • Camarillo, CA 93010 • Phone: 805/484-2806 • TWX: 910-336-1297

SEE OUR COMPLETE PRODUCT LISTING IN EEM & GOLDBOOK

The Evolution of FORTH, an Unusual Language

Charles H Moore
FORTH Inc
2309 Pacific Coast Hwy
Hermosa Beach CA 90254

Introduction

When I invented FORTH about 10 years ago, my goal was simply to make myself a more productive programmer. When I first worked with computers at MIT and Stanford in the early 1960s, I figured that in 40 years a very good programmer could write forty programs. And I wanted to write *more* programs than that. There were things out in the world to be done, and I wanted a tool to help me do them. As I worked on programs that ranged from satellite orbits to chromatography to business systems, I developed FORTH in line with my overall goal. For several years now, I have been able to work at ten times my original rate.

As I began thinking of rather drastic improvements to programs, I think I was arrogant. I wanted to do things my way. I was not convinced that I should not be permitted to, and I was a bit hard to get along with. The arrogance was necessary because I felt insecure. I was promoting ideas that everyone said were wrong and that I thought were right. But, if I were right, that meant that all the

other people would have been wrong, and there were many more of them than me. And it took a lot of arrogance to persist in the face of massive disinterest.

FORTH is a polarizing concept. There are people who love it and people who hate it. It's just like religion and politics. If you want to start an argument, say, "Boy, FORTH's really a great language."

This is partly because FORTH is an amplifier. A good programmer can do a fantastic job with FORTH; a bad programmer can do a disastrous job. I have seen very bad FORTH code and have been unable to explain to the author exactly why it was bad. There are some visible characteristics of good FORTH, such as very short definitions (many of them). Bad FORTH often takes the form of one definition per block—big, long, and dense. It is quite apparent, but difficult to explain, why or how a FORTH program is bad.

BASIC and FORTRAN are less sensitive to the quality of the programmer. I was a good FORTRAN programmer; I thought that I was doing the best job possible with FORTRAN, but it was not much better than what everybody else was doing. In this sense, FORTH is an elitist language.

On the other hand, I think that FORTH is a language that a grade school child can learn to use quite effectively, if it is presented in bite-

size pieces with the proper motivation.

FORTH is the first language that has come up from the grass roots. It is the first language that has been honed against the rock of experience before being standardized. I hesitate to say it is perfect; I will say that if you take anything away from FORTH, it is not FORTH any longer—the basic components are all essential to the viability of the language.

History

What might be called the prehistory of the FORTH language goes back much further than 10 years. The first element of FORTH to exist was the text interpreter, shown in listing 1. This early version, programmed in ALGOL at the Stanford Linear Accelerator Center in the early 1960s, was part of a program called TRANSPORT, which designed electron-beam transport systems. Besides the text interpreter, this print-out also shows an early version of the dictionary. The influence of LISP is evident in the indivisible entity (which in FORTH is called a *word*) named ATOM. As the interpreter reads a word from a punched card, it executes the associated routine, as for DRIFT in this example. The style resembles that of modern FORTH: there is no limit on the length of a word, as you can see by the length of the word SOLENOID, but only the

About the Author

Charles H Moore is Chairman of the Board of FORTH Inc, a firm created in 1973 to provide application programming services and packaged FORTH systems. This article is adapted from a speech delivered at the FORTH Convention held in San Francisco in October 1979.

first characters are significant and words are separated by spaces.

Other very early concepts have either changed in form or have evolved dramatically. In listing 2, the word that has become { : } (colon) in modern FORTH is called DEFINE, while END has become { ; }

(semicolon). This listing also shows stack operators being defined. As an example of a concept that has evolved, consider the dictionary being sealed by the word SEAL and broken by the word BREAK. Such sealing and breaking has since been replaced by the idea of vocabularies.

Listing 1: An early version of the FORTH text interpreter (written in ALGOL).

```
IF ATOM="DRIFT" THEN DRIFT
ELSE IF ATOM="QUAD" THEN QUAD
ELSE IF ATOM="BEND" THEN BEND
ELSE IF ATOM="FACE" THEN FACE(-1)
ELSE IF ATOM="ROTATE" THEN ROTATE
ELSE IF ATOM="SOLENO" THEN SOLENOID
ELSE IF ATOM="SEX" THEN SEX
ELSE IF ATOM="ACC" THEN ACC

ELSE IF ATOM="MATRIX" THEN BEGIN IF NOT FITTING THEN BEGIN
  REAL A;
  WRITE1(3,0,0,CORE(S)); LINE(-(8+42*(ORDER-1)));
  FOR J-1 STEP 1 UNTIL 6 DO BEGIN
    FOR K-1 STEP 1 UNTIL 6 DO WRITE1(2,8,R1(J,K)*UNIT[K]/UNIT[J],2);
    LINE(0) END;
  IF ORDER=2 THEN FOR C-1 STEP 1 UNTIL 6 DO BEGIN
```

Listing 2: An early version of the FORTH words { : } (called DEFINE here) and { ; } (called END here).

```
"- "OPEN DEFINE MINUS + END
SEAL "< "OPEN DEFINE - < END BREAK
"NOT "OPEN DEFINE MINUS 1+ END
"> "OPEN DEFINE .< END
"AND "OPEN DEFINE x END
"OR "OPEN DEFINE NOT .NOT AND NOT END
" T 1 1 "REAL DECLARE
"= "OPEN DEFINE T- ; DUP T< . T> OR NOT END
"≠ "OPEN DEFINE = NOT END
"≤ "OPEN DEFINE > NOT END
"≥ "OPEN DEFINE < NOT END
"DUMP "OPEN DEFINE NAME 10 "ALPHA WRITE; 3 10 "REAL WRITE 0 LINE END
```

Listing 3: Another prototype of the FORTH text editor, again in ALGOL. In this listing, the word ATOM (the predecessor of the basic unit in FORTH, the word) has been replaced by the word W.

```
120 CYCLE; FILL OUTPUT WITH BUFFER[1],BUFFER[2];
1 WHILE WORD NEQ "END " DO
2 IF W=GM1 THEN REPLY("OK ")
3 ELSE IF NUMERIC THEN L:=MIN(W-1,$OF)
4 ELSE IF W="+ " THEN L:=MIN(L+WORD,EOF)
5 ELSE IF W="- " THEN L:=MAX(L-WORD,0)
6 ELSE IF W="T " THEN BEGIN
7 IF WORD=GM1 THEN W:=1; W:=MIN(L+W-1,EOF);
8 FOR L:=L STEP 1 UNTIL W DO BEGIN
9 POSITION; TYPE END; L:=L-1 END
130 ELSE IF W="R " THEN BEGIN
1 POSITION; REPLACE END
2 ELSE IF W="A " THEN BEGIN
3 L:=EOF:=EOF+1; REPLACE END
4 ELSE IF W="I " OR W="D " THEN BEGIN
5 IF NOT RECOPY THEN BEGIN
6 RECOPY:=TRUE; REWIND(CARD) END;
7 POSITION; IF W="I " THEN BEGIN
8 PLACE; REPLACE END
9 ELSE BEGIN EMPTY:=TRUE; IF WORD NEQ GM1 THEN BEGIN
140 L:=MIN(L+W-1,EOF); SPACE(CARD,L-L0+1); L0:=L+1
1 END END END
```

Listing 3 shows another prototype in ALGOL, this time of a FORTH text editor. Here ATOM has become W and I am looking up plus, minus, and the commands T, R, A, and I, to edit a deck program.

Another method of implementing a dictionary is shown in listing 4. I am looking up the words in a conditional statement and setting NEXT, the key routine of modern FORTH's address interpreter, to the index.

Listing 5 shows an early implementation of a stack. Since it is written in BALGOL, which allows assignment statements inside other statements, I could replace STACK[J] with [J+1] in order to push items onto the stack. I did this so that I could manipulate parameters that were interpreted from the card deck as arguments to the routines. When I wanted, for instance, to convert angular measure from one unit to another, this added the ability to use arithmetic operators.

From Stanford I moved to the East Coast, where I programmed on a free-lance basis for several years. Some of you probably remember that, in the 1960s, a programmer at a typical computer center needed to learn about nineteen languages in order to function adequately: JCL (Job Control Language); languages to control utilities and facilities, such as the linking loader; assembly language and the assembler's control language; plus several high-level languages and the methods for controlling their compilers.

Listing 6 shows two of these languages, a PL/I program and the JCL necessary to run it. Note the obvious difference in syntaxes. FORTH developed in response to such conditions. In terms of modern FORTH, the importance of this example lies in the use of NEXT as a procedure that goes off to get the next word and do something with it.

Listing 7 shows a version of FORTH coded for the IBM System/360 with the routines PUSH and POP, which executed in about 15 μs. They include stack limit checking, which doubled the cost and was one of the things that led me to believe that execution-time stack checking is not desirable. This was coded in a macroassembler that did not have stack operations, which led to the deck full of statements like L19

Listing 4: An early version of the FORTH dictionary.

```
8 PROCEDURE RELEVANCE; BEGIN REAL T,K0;
9 J:=0; I:= -1; WHILE WORD NEQ "END " DO
180 IF W="" = " THEN NEXT:=3
1 ELSE IF W="GT " THEN NEXT:=4
2 ELSE IF W="LT " THEN NEXT:=5
3 ELSE IF W="NOT " THEN NEXT:=6
4 ELSE IF W="AND " THEN NEXT:=7
5 ELSE IF W="OR " THEN NEXT:=8
6 ELSE IF W="+ " THEN NEXT:=9
7 ELSE IF W="- " THEN NEXT:=10
8 ELSE IF W="* " THEN NEXT:=11
9 ELSE IF W="/" THEN NEXT:=12
190 ELSE IF K0:=SEARCH1(W) GEQ 0 THEN BEGIN
1 NEXT:=1; NEXT:=K:=K0 END
2 ELSE BEGIN
3 NEXT:=2;
4 IF BASE[K]=" " THEN NEXT:=WORDS[0]
5 ELSE NEXT:=W END;
6 NEXT:=0 END;
```

Listing 5: An early implementation of the FORTH stack, written in BALGOL.

```
7 BOOLEAN PROCEDURE RELEVANT; BEGIN
8 I:=J:= -1; STACK[0]:=1; DO CASE NEXT OF BEGIN
9 J:= -1;
210 STACK[J:=J+1]:=CONTENT;
1 STACK[J:=J+1]:=NEXT;
2 STACK[J:=J-1]:=REAL(STACK[J]=STACK[J+1]);
3 STACK[J:=J-1]:=REAL(STACK[J] GTR STACK[J+1]);
4 STACK[J:=J-1]:=REAL(STACK[J] LSS STACK[J+1]);
5 STACK[J]:=REAL(NOT BOOLEAN(STACK[J]));
6 STACK[J:=J-1]:=REAL(BOOLEAN(STACK[J]) AND BOOLEAN(STACK[J+1]));
7 STACK[J:=J-1]:=REAL(BOOLEAN(STACK[J]) OR BOOLEAN(STACK[J+1]));
8 STACK[J:=J-1]:=STACK[J]+STACK[J+1];
9 STACK[J:=J-1]:=STACK[J]-STACK[J+1];
220 STACK[J:=J-1]:=STACK[J]×STACK[J+1];
1 STACK[J:=J-1]:=STACK[J]/STACK[J+1];
2 END UNTIL J LSS 0;
3 RELEVANT:=BOOLEAN(STACK[0]) END;
```

DC AL2(*-L18), which gave me a link from L19 to the previous label. It worked but it was not pleasant.

Listing 8 shows a similar routine, this time coded in COBOL. I am setting up a table of identified words that will be interpreted from an input stream. Since COBOL does not allow parameters for subroutines, it is awkward to do anything meaningful.

New Concepts

About this time, I began to think of defining a word that would define other words; and at that time, this idea was staggering. For example, { ;CODE } was a very esoteric word. I explained it to people, but I could not express the potential I thought it had.

It took time to find out exactly what { ;CODE } should do (it specified the code to be executed for a previously defined word). I do not have the records, but I think the initial code for { ;CODE } was three or four lines long; to simplify that code

was one of the driving forces behind the address interpreter—to make it possible to code { ;CODE } cleanly. This had implications as to what registers should be available.

The fact that W should be saved in a register for defining words led to *indirect*, rather than direct, threaded code. That was the most complicated concept I had coded in this evolving program—probably deserving of a patent in its own right.

A little bit later, it seemed that there ought to be an analog of { ;CODE } that specified the code to be *interpreted* when you executed a word. It seemed the natural balance, but when the idea first arose, I did not have the foggiest notion of what to do or what the implementation should be. The first definition of this analog, called { ;: } (semicolon-colon), required three or four lines of code. It had to do what { ;CODE } did, and then more.

Out of that came the distinction between compile-time action and

execute-time action. It was convenient for words to be coded to act this way, but it was expensive. It required not only the address of the code to be executed, but the address of the code to be interpreted, as well as the parameter to be supplied to the code being interpreted so you could do something useful.

Late in the 1960s I went to work for Mohasco Industries, where I put something strongly resembling FORTH on a Burroughs 5500, cross-compiled to the 5500 from an IBM 1130. (There is no assembler on the 5500; there is a dialect of ALGOL called SBOL that Burroughs used to compile operating systems, not

available to users.) Listing 9 shows the code definitions of stack operations on the 5500, which was a stack-oriented processor at a time when stack machines were not popular. The names of some FORTH stack operators stem from that machine's operations; see, for example, DUP . The symbol ϵ stands for CODE and distinguishes the assembler's OR from the FORTH OR . (Vocabularies were not yet available.)

Listing 10 gives an example of FIND (a dictionary search routine) coded for the 5500. Notice the word SCRAMBLE , a colon definition making a hashed search. Apparently I had eight threads to the dictionary here, a

concept we added back to FORTH when we developed polyFORTH last year.

FORTH and the IBM 1130

At Mohasco I also worked directly on an IBM 1130 interfaced with an IBM 2250 graphics display. The 1130 was a very important computer; it had the first cartridge disk, as well as a card reader, a card punch (as backup for the disk), and a console typewriter. The 1130 let the programmer, for the first time, totally control the computer interactively.

FORTH first appeared as an entity on that 1130. It was called F-O-R-T-H, a five-letter abbreviation of FOURTH, standing for fourth-generation computer language. That was the day, you may remember, of third-generation computers and I was going to leapfrog. But because FORTH ran on the 1130 (which permitted only five-character identifiers), the name was shortened.

What came out of the 1130 was a cross-assembler that assembled the instructions, which were then to be executed by the 2250. I think the 2250 had its own memory, and these things had to be programmed carefully. What I accomplished was that the 1130 in FORTRAN in 32 K bytes could draw pictures on the 2250, fairly slowly; and FORTH, in 8 K bytes, could draw three-dimensional moving pictures on the 2250—but it could do that only if every cycle was accounted for and if the utmost was squeezed out. That is why FORTRAN had to go—I required an assembler and could not do an impressive enough job with FORTRAN.

But high-level or colon definitions were not yet compiled—the compiler came much later. The text was stored in the body of the definition, and the text interpreter reinterpreted the text in order to discover what it was to do. This contradicts the efficiency of the language, but I had big words that put up pictures and I did not have to interpret too much. The cleverness was limited to squeezing out extraneous blanks as a compression medium. I am told that this is the way that BASIC acts today in many instances.

This machine had a disk drive, and I am almost certain that the word BLOCK existed in order to access

Listing 6: The NEXT procedure in PL/I and its associated JCL (Job Control Language) statements (lines 1 thru 8).

```

1 //UTILITY JOB SYSTEM,OVERHEAD
2 // EXEC PGM =IEBUPDTE,PARM =NEW
3 //SYSPRINT DD SYSOUT =A
4 //SYSUT2 DD DSNAME =OUTLIB,UNIT = 2314,DISP =(NEW,KEEP),
5 // VOLUME =SER =MOORE,SPACE =(TRK,(100,,10)),
6 // DCB =(RECFM =F,LRECL =80,BLKSIZE =80)
7 //SYSIN DD DATA
8 ./ ADD NAME =WORD,LEVEL =00,SOURCE =0,LIST =ALL
9 DECLARE KEYBOARD STREAM INPUT,PRINTER STREAM OUTPUT PRINT;
10 NEXT: PROCEDURE CHARACTER(4);
1 DECLARE (1 TEXT CHARACTER(81) INITIAL((81)" "),
2 2 C(81) CHARACTER(1), I INITIAL(81),W CHARACTER(4),
3 WORD CHARACTER(32) VARYING BASED(P),P,NUMERIC BIT(1)) EXTERNAL;
4 DO WHILE C(I) = " "; I=I+1;
5 IF I=82 THEN BEGIN; I=1;
6 READ FILE(KEYBOARD) INTO(TEXT); END; END;
7 P = ADDR(C(I));
8 IF C(I) = "-" OR C(I) = "." OR "0" LE C(I) THEN BEGIN; NUMERIC = "1"B;
9 IF C(I) NOT = "." THEN DO I=I+1 BY 1 WHILE "0" LE C(I); END;
20 IF C(I) = "." THEN DO I=I+1 BY 1 WHILE "0" LE C(I); END; END;
1 ELSE DO; NUMERIC = "0"B;
2 IF "A" LE C(I) THEN DO I=I+1 BY 1 WHILE "A" LE C(I) OR C(I) = "-";
3 END; ELSE I=I+1; END;
4 W = WORD; RETURN(W);

```

Listing 7: The FORTH words PUSH and POP written in IBM 360 assembly language.

```

0056          830 L18 DC AL2(*-L17)
              831 NAME 3,X'445550',0 DUP
03445550     832+ DC AL1(3),X'445550'
00          833+ DC X'0'
              834+ ORG *-2-V0
              835+ DS 0H
              836+ ORG *+V0+1
              837+ DC AL1(0*X'40'+X'40'),AL2(4)
400004       00014 838 PUSH A SP,MFOUR COSTS 15 US
5ACO 6014    00000 839 ST T,0,(SP)
5040 C000    00000 840 CR SP,DP
19CB        841 BCR 2,NEXT BHR
0729        842 B ABORT
47F0 667C   0067C 843 L19 DC AL2(*-L18)
001A        844 DC AL1(4),X'44D2CF50',X'40',AL2(8) DROP
0444D2CF50400008
41C0 C004    00004 845 LA SP,4,(SP)
5840 C004    00004 846 POP L T,4,(SP) COSTS 21 US
41C0 C004    00004 847 LA SP,4,(SP)
59C0 602C   0002C 848 C SP, SP00
07C9        849 BCR 12,NEXT BNHR
47F0 667C   0067C 850 B ABORT

```


records off the disk. I do remember that I had to use the FORTRAN I/O (input/output) package and that it would not put the blocks where I wanted them; it put the blocks where it wanted them, and I had to pick them up and move them into my buffers.

At Mohasco I also implemented FORTH on a Univac 1108, interfacing it with their COBOL compiler. Listing 11 displays a set of record descriptions in a Dun and Bradstreet reference file (for looking up bad debts). The layout shows named fields followed by the number of bytes allocated.

The Mohasco programs mark the transition point between something that could be called FORTH and something that could not. All the essential features except the compiler were present by 1968.

The First Modern FORTHS

The first modern FORTH was coded in FORTRAN. Shortly thereafter it was recoded in assembler. Much later it was coded in FORTH. It took a long time before I thought that FORTH was complete enough to code itself. The first thing to be added to what had already existed was the return stack. That was an important development; the recognition that there had to be exactly two stacks, no more, no less.

The next thing to be added was even more important—the *full-fledged dictionary*, that is, the dictionary in the form of a linked list. Up until then, flags had been set or computed GO TOs had been executed to provide some mechanism for associating a subroutine with a word. The replacement of all that by a code file containing the address of the routine made an incredibly fast way of implementing a word once it was identified.

The first use of modern FORTH occurred when it was written for a Honeywell H316 at the NRAO (National Radio Astronomy Observatory). In 1971 I was hired by George Conant to write a radio-telescope data-acquisition program: that led to the next step, the compiler. This meant the recognition that, rather than reinterpret a string of text, words could be compiled and an average of 5 characters per word could be replaced by 2 bytes per word. This gave a compression factor

Listing 8: A structured table routine, in COBOL.

```

1      MOVE "CONFIGURATION" TO IDENTIFY(4);
2      MOVE "DATA" TO IDENTIFY(5);
3      MOVE "FILE" TO IDENTIFY(6);
4      MOVE "FD" TO IDENTIFY(7);
5      MOVE "MD" TO IDENTIFY(8);
6      MOVE "SD" TO IDENTIFY(9);
7      MOVE "WORKING-STORAGE" TO IDENTIFY(10);
8      MOVE "CONSTANT" TO IDENTIFY(11);
9      MOVE "PROCEDURE" TO IDENTIFY(12);
70     MOVE "INPUT-OUTPUT" TO IDENTIFY(13);

```

Listing 9: Code definitions of FORTH stack operations on the Burroughs 5500, written in SBOL.

```

LIST
0001 ( 'PRIMITIVES' 26 LAST = 30 SIZE = )
0002 ⚡ = _S RETURN
0003 ⚡ @ <SD RETURN
0004 ⚡ + V 241, RETURN
0005
0006 ⚡ OR ⚡OR RETURN
0007 ⚡ AND ⚡AND RETURN
0008 ⚡ NOT 115, RETURN
0009 ⚡ DUP ⚡DUP RETURN
000A ⚡ SWAP ⚡SWAP RETURN
000B ⚡ DROP ⚡DROP RETURN
000C ⚡ + +1 RETURN
000D ⚡ - -1 RETURN
000E ⚡ MINUS ⚡MINUS RETURN
000F ⚡ * *1 RETURN
0010 ⚡ / /1 RETURN
0011 ⚡ MOD ⚡MOD RETURN

```

Listing 10: A dictionary search routine, FIND, written for the Burroughs 5500.

```

0013 ⚡SM ⚡FIND SCRAMBLE <SD ⚡DUP
0014 41 >A 41 >B ⚡BEGIN V <U 1771, ⚡IF
0015 ⚡BEGIN V0 <U 1771, ⚡IF
0016 1 <L RESULT
0017 ⚡THEN __ADDR ⚡DUP 1 <L <S
0018 OS WORD <U ⚡EQUAL ⚡IF
0019 V1 __U OS RESULT
001A ⚡THEN ⚡DUP <SD ⚡BACK
001B ⚡THEN GET ⚡BACK
001C : FIND TOP ⚡FIND ⚡IF UR <UD ⚡B ⚡THEN;

```

Listing 11: Prototype of a file layout, running under FORTH on a Univac 1108. This version of FORTH was written in COBOL.

```

3 DBI DBI/MOORE 33 33
4 DUNS 8 NAME 24 STREET 19 CITY 15 STATE 4 ZIP 5
5 PHONE 10 BORN 3 PRODUCT 19 OFFICER 24 SIC 4 SIC1 4 SIC2 4
6 SIC3 4 SIC4 4 SIC5 4 TOTAL 5.0 EMPL 5.0 WORTH 9.0 SALES 9.0 MFG 1
7 SUBS 1 HDQ 1 HEAD 8 PARENT 8 MAIL 19 CITY1 15 STATE1 4
8 NAME1 19
9 END

```

of 2 or 3, not drastic but appreciable. But execution speed would be much faster. Again I asked myself, as I had done when I first began modifying programs: if it was that easy, why hadn't anyone else done it? It took me a long time to convince myself that you could compile anything and everything.

Interrupts came around this time. It

was important to utilize the interrupt capability of the computer, but it had not been done by me before that—I did not know anything about interrupts. I/O, however, was not yet interrupt-driven. Interrupts were available for the application if it wanted them—FORTH did not bother.

The multiprogrammer came along

a couple of years later when we developed an improved version of the system for NRAO's telescope at Kitt Peak. This computer was a PDP-11; the multiprogrammer had four tasks. Input was still not interrupt-driven, which was unfortunate.

The Second FORTH Programmer

Ten years ago there was one FORTH programmer, me. The second FORTH programmer, Elizabeth Rather, came along in 1971. That is quite a quantum jump, from one to two; the next step was four (the next two came out of Kitt Peak National Observatory); the growth can be traced from there to the several thousand today.

The first FORTH user was Ned

Conklin, head of the NRAO station at Kitt Peak, Arizona. NRAO runs a millimeter-wave radio telescope that is in great demand by observers, in part because it is responsible over the last 10 years for discovering half of the interstellar molecules that are known to exist. FORTH is still running on that telescope at Kitt Peak and on a lot of other telescopes.

Given interest from other astronomers, a few believers split off from NRAO in 1973 and formed FORTH Inc. We were deluged by requests for FORTH systems from astronomers and went into business to try to exploit that market. It would still be our principal line of business today except that there are so few new telescopes in the world that you

cannot support a company on that market.

We developed miniFORTH™ (FORTH on minicomputers) with the idea of having a programming tool. An important implementation of the tool came when we put an LSI-11 and FORTH into a suitcase. I think I became the first computer-aided programmer—computer-aided in that I had my computer and took it around with me. I talked to my computer, my computer talked to your computer, and we could communicate much more efficiently than I could communicate directly with your computer before it could run FORTH. Using this tool, we have put FORTH on many computers.

We added the feature of interrupt-driven I/O when FORTH Inc produced its first multiterminal system. It did not speed things up particularly from the user's point of view, but it *did* prevent any loss of characters when several people were typing at the same time. You did not have to look quickly to get the character before the next one came along. They were all buffered and waiting for you, which is an important distinction for multiprogrammed systems.

Data-base management came along at this time. It has been extensively changed, just as FORTH has. But fundamentally, nothing has changed. The concept of files, records, fields, and relational pointers that polyFORTH™ offers dates back from 1974 or so—years and years ago. Listing 12 shows a recent application of the FORTH Inc data-base management system.

With microFORTH™ in 1976 came the first version of our current target compilers. They are very complex things, much more so than I expected them to be. At about the same time, we worked out the current implementation of DOES> .

This new form of { ;: } does not require the address of the code to be interpreted. Since that is supplied by a different mechanism, the parameter can occupy the parameter field as it is supposed to. You can "tick" it and change its value, which is nice. [The FORTH word { ' } (called "tick" above) places the address of the word that follows it onto the stack....GW] But we save 2 bytes for every DOES> word, 2 bytes for very common words—and for 3 years, we did

Listing 12: Field and record layouts for a recent FORTH Inc data-base management system.

64 LIST

```
0 ( GLOSSARY FILE)
1 2 ( LINK) 12 BYTES WORD 12 BYTES VOC
2 NUMBER SOURCE NUMBER STACKS 70 BYTES PHRASE
3 210 FILLER ( 4 LINES) 32 FILLER ( 340 B/R, 3/BLOCK) DROP
4 2 24 BYTES WORD+VOC DROP
```

Hard Disk Made Easy

Now you can move up to hard disk trouble free. Just select the XCOMP X/S series controller for your disk drive: SMD, Cartridge drive, 8 inch disk bus or Shugart® SA1000. Our complete package, including first class support software, will get you up and running fast. And the cost will be less than you would expect. We specialize in getting OEM's into hard disk systems. Our customers include the most successful companies in the microcomputer world.

Move up to hard disk the easy way. Call XCOMP—we'll get you going with hard disk right now.

XCOMP
INCORPORATED

9915A Businesspark Avenue,
San Diego, CA 92131
(714) 271-8730

not realize that we had missed the optimum by so much.

I know no way of speeding this process from initial thought to development, except to let a certain amount of time pass. We could sit, we did sit and debate this thing endlessly, and we missed the obvious.

I think that completes the capabilities that I think of as FORTH today. You see how they dribbled in—at no point did I sit down to design a programming language. I solved the problems as they arose. When demands for improved performance came along, I would sit and worry and come up with a way of providing improved performance.

polyFORTH is a condensation of everything that we at FORTH Inc have learned in the last 10 years of developing FORTH. I think it is a very good package. I foresee no fundamental changes in the design of the language except for accommodation to FORTH standards, which are becoming increasingly important.

Implementations of FORTH

I would like to review the implementations of FORTH of which I am aware. It is actually a tour through the history of computers and it is fascinating that this could all have happened in 10 years.

FORTH has been programmed in FORTRAN, ALGOL, PL/I, COBOL, assembler, and FORTH; and I am sure some of you can come up with other languages with the same history. My list is strictly personal.

FORTH has been implemented on the Burroughs 5500; the IBM 1130; the Univac 1108; the Honeywell 316; the IBM 360; the Data General Nova; the HP 2100 (not by me but by Paul Scott at Kitt Peak); the PDP-10 and PDP-11 (by Marty Ewing at the California Institute of Technology); the PDP-11 (by FORTH Inc); the Varian 620; the Mod-Comp II; the GA SPC-16; the CDC-6400 (by Kitt Peak); the PDP-8; the IV-Phase; the Computer Automation LSI-4; the RCA 1802; the Honeywell Level 6; the IBM Series 1; the Interdata; the 6800; the 8080; the 8086; the TI-9900; and soon the 68000, the Z8000, the 6809, and a Child Inc computer. Some independent groups have 6502s, ILLIAC, and others running FORTH. I raise the question—is it the case that FORTH has been put on

every computer that exists?

Some people think FORTH ought to be machine independent, but that premise is wrong. The equivalence is FORTH—each machine requires meticulous attention to its individual characteristics. You must use all the hardware capabilities of each machine and must then work to force it into the mold specified by FORTH's virtual machine.

For example, we put a subset of FORTH on an SMS-300 microcomputer. It had only eight instructions. The internal characteristics of every

At no point did I sit down to design a programming language. I solved the problems as they arose.

machine can and must be exploited. You do not need any particular number of registers or stacks or anything. All can be simulated, but if you neglect the abilities of the machine, you can end up a factor of 2 down in performance from where you might otherwise be.

FORTH-in-Hardware Computers

The first FORTH computer I know of was built at Jodrell Bank in England around 1973. It is a redesign of an English Ferranti computer that went out of production. The observatory at Jodrell Bank was going to build their own bit-slice version; they discovered FORTH about the same time, modified the instruction set to accommodate FORTH, and built what I am told is a very fast FORTH computer. I have never seen it, but have talked to its competent designer, John Davies, who is one of the early FORTH enthusiasts.

In 1973, before Dean Sanderson came to FORTH Inc to develop microFORTH, he had a FORTH computer at a company called General Logic. It qualifies as a FORTH computer because it has a FORTH instruction. And there is a story there. Dean showed me his instruction set, and there was this funny instruction that I could not see any reason for—I figured it was some kind of no-op or catchall or

something; it had the weirdest properties, and it could not possibly be useful. It was NEXT. It was a one-instruction NEXT which was beautiful. And it was a very simple modification (this was a bit-slice computer) to the instruction set—a few wires here and there—and that is the first time I saw a FORTH computer, if you will. I call it a FORTH computer because it had the ability to change itself from an ordinary computer into a FORTH computer.

I think that hardware today is in the same shape as software was 20 years ago. No offense, but it is time that the hardware people learned something about software. There is an order or two of magnitude improvement in performance possible with existing technology. We do not need picosecond computers to make really substantial improvements in execution speed. Faced with that realization, there is no point in trying to optimize the software any further until we have taken the first crack at the hardware. The hardware redesign has to be as complete as the software redesign was. The standard microprocessors did not have FORTH in mind. Those minicomputers that can be microprogrammed cannot be microprogrammed well enough to even be worth doing. The improvements available are much greater than you can achieve by these half measures.

I have built a small FORTH computer. The design changes as fast as the chips can be plugged into the board. But it is not difficult to do. Here are the characteristics of a FORTH computer:

- It does not need a lot of memory (16 K bytes is about right—half programmable read-only memory, half user programmable memory, maybe).
- It does not need a lot of I/O ports; in fact, it does not need any I/O ports except for the application requirements.
- A serial line and interface to a disk drive are useful but not required.

We have put FORTH on an 8080-based machine with a virtual disk in memory, enough memory to hold eight blocks. The system is quite viable and has no particular problem with system crashes. Bubble

memories are coming. A FORTH computer does not need much mass storage; 100 K bytes are adequate, and 250 K bytes are plenty. The fact that FORTH can exist quite happily on a machine that is very small by contemporary standards should be exploited.

Organizations

Finally, I would like to run through the history of the organizations that have been involved with FORTH. They have formed another thread of the tapestry. It began with Mohasco, of course, followed by NRAO and

Kitt Peak National Observatory; then came FORTH Inc.

The next step was probably DECUS (Digital Equipment Computer Users' Group). Marty Ewing gave his PDP-11 FORTH system to DECUS. FORTH Inc was not sure whether free FORTHS floating around was a good idea at the time. But it turned out that a lot of people were exposed to FORTH who otherwise would not have been.

Cybek came along and provided an entry into the business-systems market. Art Gravina, the president of Cybek, is the person who designed

our data-base management system. He provided us the opportunity to do commercial systems and the ability to handle ten times as many terminals as he could with the BASIC program that preceded it.

In about 1976, a committee of the International Astronomical Union met and adopted FORTH as a standard language. That was a boost in the world of astronomy, although the world of astronomy was no longer the major driving force in the popularity of FORTH.

I think EFUG (the European FORTH Users' Group) came along about that time (1976). It turned out to our surprise that Europe was a hotbed of FORTH activity that we were largely unaware of (and perhaps still are, in that we are not involved in that world and do not appreciate the level of interest). An international FORTH Standards Team probably grew from their first meetings. A couple of years later, the FORTH Interest Group started. Now we have FORML—FORTH Modification Laboratory, an idea-generating organization.

Conclusion

The tendency seems to be for people to organize themselves in groups. Some of these groups are companies, others are associations. It looks like FORTH is going to be a communal activity in that sense—that it will grow from the work of unstructured clusterings of like-minded people. The suggestion is that this whole world of FORTH is going to be quite disorganized, uncentralized, and uncontrollable. It's not bad, perhaps it's good.

My view of the future is more unsettled today than it has been for years; promising, confusing, perplexing. The implications are perhaps as staggering now as they were 20 years ago. The promise of realization is much higher. My original goal was to write more than forty programs in my life. I think I have increased my throughput by a factor of 10. I do not think that that throughput is program-language limited any longer. So I have accomplished what I set out to do: I have a tool that is very effective in my hands. It seems it is very effective in others' hands as well. I am happy and proud that this is true.

No typing skills required

It's easier and more accurate to enter alphanumeric data with a BIT PAD than a keyboard. Now anyone can...

- Enter whole lines of characters with a single stroke.
- Enter data directly from business forms by simply checking a box.
- Enter variable alphanumeric data from a menu keyboard.

Take a printed form—price list, order form, loan or insurance application, laboratory request—lay it on the BIT PAD tablet and touch the pertinent items with the pen. The information is entered directly into your data processing system.

Plus, the BIT PAD does even more.

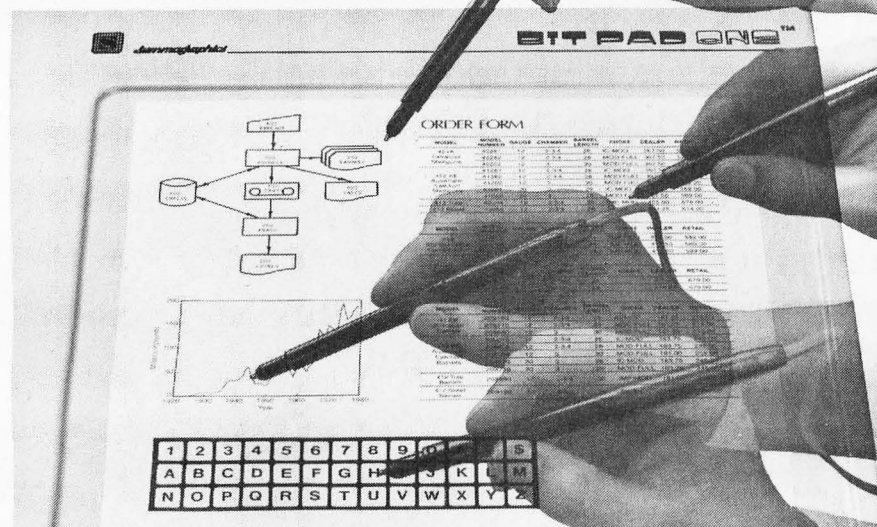
Try to describe a fluctuating business trend to your computer through a keyboard. With BIT PAD you simply trace the trend with the pen. Special keyboard menus can be created by the user to enter high level languages, foreign languages or special symbols.

Before you order any kind of data entry equipment, ask Summagraphics to give you the full story on the BIT PAD ONE.

Summagraphics Corporation, 35 Brentwood Avenue, Fairfield, Connecticut 06430; or call Marketing Department, Peripheral Products (203) 384-1344.



The BIT PAD™ alternative to keyboard data entry



Components of FORTH

FORTH is characterized by five major elements: dictionary, stack, interpreters, assembler, and virtual memory. Although not one of these is unique to FORTH, their interaction in FORTH produces a synergistic effect that creates a programming system of unexpected power and flexibility.

- **Dictionary:** The resident FORTH system is organized into a dictionary that occupies almost all of program memory. The dictionary is a threaded list of variable-length items, each of which defines a word of the vocabulary. The actual content of each definition depends on the type of word: noun, verb, etc. The dictionary is extensible, growing toward high memory. In a multiterminal system, terminal tasks may have private dictionaries that are connected in a hierarchical tree structure.
- **Stack:** Two push-down stacks (last-in, first-out, or LIFO, lists) are maintained for each multiprogrammed task in the system. These provide the primary communication between routines as well as an efficient mechanism for controlling logical flow. A stack normally contains items one computer word long, which may be addresses, numbers, or other objects. Stacks, which are of indefinite size, grow toward low memory.
- **Interpreters:** FORTH is fundamentally an interpretive system, meaning that program execution is controlled by data items rather than by machine code. It is a common assumption that interpreters are severely wasteful of processor time; this is avoided in FORTH by maintaining two levels of interpretation.

The first of these is the text interpreter, also known as the outer interpreter. It works in a conventional manner, parsing text strings that come from terminals or mass storage and looking up each word in the dictionary. When a word is found in the dictionary, it is executed (unless the task is in compile mode) by invoking the address interpreter.

The address interpreter (also known as the inner interpreter) interprets strings of absolute memory addresses by executing the definition pointed to by each. Most dictionary definitions contain addresses of previously defined words that are to be executed by this interpreter. This level of interpretation requires no dictionary search since these words have already been compiled by the text interpreter, which generated the absolute addresses.

The address interpreter has several important properties. First, it is fast. Indeed, on some computers it executes only one instruction for each word, in addition to the code implied by the word itself. Second, it interprets compact definitions. Each word referenced in a definition compiles a single memory location. Finally, the definitions are machine independent because the definition of one word in terms of others does not depend upon the computer that interprets the definitions.

- **Assembler:** FORTH includes a resident assembler, which allows the programmer to define words that will cause specified machine instructions to be executed. This type of definition is necessary to perform device-dependent input and output operations, to implement elementary operations, and to do highly time-critical processing.
- **Virtual memory:** The final key element of FORTH is its blocks: fixed-length segments of disk space that may contain program text or data. A number of buffers are provided in memory; blocks are read into them automatically when referenced. If a block is modified in memory, it is automatically replaced on disk. Explicit read and write operations, therefore, are not required; programmers may presume that program text or data is in memory whenever it is referenced.

[The above paragraphs present a concise overview of FORTH as a language; the following paragraphs describe features of a FORTH Inc product, polyFORTH...GW]

The standard polyFORTH system utilities include the following:

- | | |
|------------------|--|
| Text editor: | Facilitates editing program source text, both by line and by character. |
| Source listings: | Prints program source listings and indexes. |
| Disk copy: | Provides for disk-to-disk copying of data file and program source files for backup purposes. |
| Disk diagnostic: | Produces a simple, read-only disk diagnostic that may be run at any time without disturbing other users. (More extensive hardware diagnostics are optional.) |

Each polyFORTH system also contains a Target Compiler™ capability; this allows the user to develop, for run-time applications only, a computer system that does not require the entire operating system. Since FORTH is an interpretive language, an interpreter must always be present; but the target compilation process creates the minimum dictionary necessary, thus allowing a program to be run with a minimum of memory overhead. Typically, this overhead is less than 1000 bytes.

Full data-base management support is available in an optional Extended File Management package. Included within its structure are the essential features of the CODASYL standard along with the characteristic speed, compactness, and flexibility of the FORTH language. Facilities include commands for file definition and formatting and for field and record descriptions, as well as several file-accessing techniques, operators for accessing individual fields by name and fields within specified files, and such utility functions as a report generator and an optional key-sort routine. ■

What Is FORTH?

A Tutorial Introduction

John S James
POB 348
Berkeley CA 94701

FORTH is a programming language with a small but fast-growing and enthusiastic user community. Though easy to learn at a terminal, it is difficult to explain abstractly because it is so different from other languages. Even advocates do not agree why it is good or how it should be used.

FORTH was developed for control applications (using a computer to run other machinery), data bases, and general business. It is least useful for big number-crunching jobs (eg: writing a matrix inversion routine), although it can link to subroutine packages written in other languages to incorporate such functions. Unlike Pascal, FORTH gives the user complete access to the machine and does not try to guard the programmer against mistakes. But its modularity and other forms of error control allow production of remarkably bug-free application programs—perhaps

more than any other language in common use. The compiler uses much less memory than Pascal does, and its programs run about equally fast. FORTH is much more interactive than most conventional implementations of Pascal. FORTH is available on most common personal computers (eg: Apple, TRS-80) and all major microprocessors (eg: 8080, 6800, 6809, 6502, PACE, LSI-11, and 9900). An international FORTH Standards Team exists, and standard systems are virtually identical among all different machines.

This article will describe what it is like to program in FORTH. A group of annotated terminal sessions, shown in listings 1 thru 10, will provide more details on the language itself.

The Philosophy of FORTH

FORTH reduces the cost of a subroutine to very little, and the whole language is built on functions that are like subroutine calls. The programmer keeps defining new words (new functions) from old ones until, finally, one of them is the whole job. Most programmers keep each definition short, usually one to three lines not counting comments. The definitions are compiled as entered and are immediately ready to run.

Because FORTH definitions are short, all possible execution paths of the definition can be tested easily. Since most functions work exactly the same when executed as commands from the terminal or when used as components in further definitions,

they can be tested immediately from the terminal. And the functions are so general that there is no sharp distinction between program and data.

Since programmers define their own operations, special application libraries of FORTH words can be developed. The new routines are integrally part of the language, so they do not need any special calling sequences, and they are immediately ready to run. Even the original words supplied with the system (there are about one hundred of them), can be redefined if desired, adapting the language for special circumstances. Also, programmers can create their own data types or operation types (eg: their own kinds of arrays or other data structures, or new classes of operations). This flexibility allows unprecedented "customization" of a language to the requirements of a particular installation or application. The finished programs are easily modifiable when requirements change because they are composed of pretested building blocks specially designed for that kind of program.

Stack and Postfix Notation

A smaller convenience of FORTH is that you do not have to do much coding when you start a new program. As soon as the system comes up, all your previous work is ready to go, just as if it were originally part of the language.

A feature that some people do not like is FORTH's use of a stack (explained below) and its *postfix notation* (also called *reverse Polish*

Acknowledgments and Availability

Listings 1 thru 7 were run on a FORTH system for the Apple II provided by Cap'n Software, POB 575, San Francisco CA 94101. The PDP-11 examples were run on a system written and distributed by the author. The 8080 example was provided by John Cassidy of the Forth Interest Group, POB 1105, San Carlos CA 94070; a similar 8080 FORTH system is available from Forthright Enterprises, POB 50911, Palo Alto CA 94303. Other members of the Forth Interest Group contributed helpful suggestions. And of course we are indebted to the inventor of FORTH, Charles Moore of FORTH Inc, 2309 Pacific Coast Hwy, Hermosa Beach CA 90254, who started it all.

Most FORTH operations communicate only through a stack.

notation or RPN). In postfix notation (a system used on most Hewlett-Packard calculators), arithmetic formulas are written with the operations after their arguments, not between them. For example, "2+3" becomes { 2 3 + } in FORTH or other postfix systems; "(4+5)*(6+7)" becomes { 4 5 + 6 7 + * }. (See explanation below.) No parentheses are needed in postfix.

Some programmers do not like postfix, and they ask, "Why doesn't someone write an algebraic-to-postfix translator for FORTH? That would be easy to do." The reason is that postfix has benefits far more important than the compiler-writer's convenience. It greatly simplifies linkage to subroutines. With postfix, you do not need any CALL statement or argument list, or any formal parameters in the subroutine. While arithmetic-formula operations (add, subtract, etc) must take either one or two arguments and return exactly one result, postfix functions can have any number of arguments or results.

In FORTH, most operations communicate only through a stack. The stack, perhaps the most important data structure in programming, is used in almost all languages, but most languages hide it from the user. In FORTH, the user controls the stack directly.

A *stack* is a pile of numbers where the last ones put in are the first ones taken out; that is, you can only remove the number that is on top of the stack. It is like a stack of trays in a restaurant; trays are conveniently added and removed only at the top. (Unfortunately, computer-science texts do not agree on terminology, and a few call the top of the stack "the bottom.")

To see how a stack works in computation, consider the expression { 2 3 + } above. In FORTH, numbers are compiled as operations which place their values onto the stack. So when the 2 is executed, it is placed on top of the stack, which then looks as follows:

2

—
—

where the dashes represent whatever data may have been on the stack before. Then after the 3 has been encountered, the stack becomes:

3

2

—
—

Then the + is executed. The 1-character word + takes two arguments from the stack (destroying them), performs the addition, and leaves the result on the stack. So the stack finally is:

5

—
—

The reader can verify that when the formula { 4 5 + 6 7 + * } is executed, the stack goes through the sequence shown in figure 1.

Now we can see why FORTH is not the best language for big number-crunching jobs. Numbers to be operated on must be moved to the stack in addition to whatever operations are to be carried out, and this extra movement slows FORTH down for this kind of computation. Once on the stack, however, arithmetic is fast (for example, single instruction execution for addition on some 16-bit machines, more for 8-bit machines). Also, FORTH can link the useful instructions of one routine and those of another in as little as one or two instruction executions (depending on machine architecture). This makes FORTH programs much faster than BASIC, usually ten times faster or more (assuming an interactive BASIC, that is—FORTH is always interactive). But a good FORTRAN

STACK		4	5		6	7	13		117	117
	-	-	-	-	-	-	-	-	-	-
OPERATION JUST PERFORMED	(BEGIN)	4	5	+	6	7	+	*	(END)	

Figure 1: Evaluation of the postfix-notation expression, { 4 5 + 6 7 + * }. Numbers are pushed onto the stack at the top. Operators (here, + and *) pop the top two entries off the stack and push the result of that operation back on the stack. For example, the first plus sign (column 4) replaces the 4 and 5 on the stack with 9, the result of the addition operation.

compiler's code may do number-crunching several times faster still.

Characteristics of FORTH Code

FORTH is a structured language (as is Pascal) in that it has no GOTO or statement labels in the language. Discussion of structured programming is outside the scope of this article, but its importance for program correctness and maintainability is recognized.

FORTH object code (ie: a compiled program) is extremely compact, even more so than machine language. The reason is that no matter how much work an operation performs, each invocation of it takes the same space in the object program—two bytes. The bigger the program, the greater the memory advantage, since the hierarchical structure of programs allows increasingly powerful and application-targeted operations to be built up. But FORTH has a relatively large run-time memory overhead, so small programs can take less total space in other languages.

[The reason that a FORTH call can be shorter than a normal machine-language subroutine call (usually three bytes) is that a FORTH program is interpreted by a FORTH interpreter (also part of the FORTH language) in much the same way that a BASIC program is interpreted by a BASIC interpreter. The "relatively large run-time memory overhead" mentioned above is the FORTH interpreter plus a core of FORTH words defined in machine language. When a FORTH program is very large, it saves enough memory in FORTH calls to make up for run-time memory overhead....

GW]

The complete FORTH system (itself largely written in FORTH) takes about 7 K bytes, and this whole system including the compiler is com-

only left in memory as a run-time package. Therefore, 16 K bytes and a floppy disk for storing source programs are sufficient hardware for an excellent FORTH system (compare this with the memory requirements of Pascal, 48 K bytes or more). When compactness is especially important, as when programs are burned into read-only memory and embedded in machinery, FORTH's compiler, terminal handler, and operation names—anything not needed to run—can be stripped out of the application program, leaving a run-time package of about 800 bytes,

instead of the usual 7 K bytes.

FORTH programming is *reentrant*; this means that different users can share the same copy of a program in memory while running at the same time. FORTH easily handles multitasking, including multiple terminals used for program development. (At present, however, most of the low-cost systems on the market are still single-user.) FORTH is *recursive*, meaning that routines can invoke themselves.

Suppose you want to link your high-level-language program to a machine-language subroutine (eg:

you may be controlling a high-speed device and need the full speed of the computer to keep up). Many languages make this linkage difficult or impossible. In FORTH, however, it is very convenient. You can type in or load from disk a machine-language routine, using a FORTH assembler, and the new routine can be executed immediately. Listing 9 shows examples for PDP-11 and for 8080.

The word CODE invokes the FORTH assembler and begins the definition of a machine-language routine. Mnemonic instructions and address-mode symbols are understood by this assembler, and the whole power of FORTH is available for address arithmetic at assembly time. FORTH assemblers use postfix notation, so op codes come *after* their addresses, not before as in conventional assemblers.

The machine-language code is generated as the definition is being entered. The completed operation works just like any other FORTH word, so the user does not need to use any special calling sequence, or even need to know which operations are defined in code and which are not. (In fact, about fifty FORTH words are written in machine language—all other words in FORTH are ultimately defined in terms of these fifty words.)

The FORTH assembler allows structured conditionals and loops at the machine-code level; it can also assemble unstructured code if desired. Users can define their own macro-instructions, use custom-made data types, etc.

In other words, the FORTH assembler allows structured programming even in machine code, and it links the resulting machine-language subroutines into the system immediately. No separate assembly and linking-loader passes are needed, and the associated file management overhead is avoided.

Some More Advantages

FORTH programs are highly transportable between different computers. Any assembly-language routines used by the program must be rewritten, but most applications do not need any assembly, and very few need more than a handful of short, critical routines. When FORTH systems have been designed for compatibility, large applications can be moved among very different

SOFTWARE FOR THE HARDWARE

We know you hardcore bit hackers will recognize the computing power derived from combining the FORTH language with the 6809, today's most advanced 8 bit microprocessor.

And we know you'll understand this machine's 16 bit math, indirect addressing and two stacks are ideally suited for implementing FORTH.

But...should anyone need further convincing that FORTH provides a new dimension in power, speed and ease of operation, consider the following:

- It's a modern, modular, structured-programming high-level compiled language.
- It's a combined interpreter, compiler, and operating system.
- It permits assembler code level control of machine, runs near speed of assembler code, and uses less memory space than assembler code.
- It increases programmer productivity and reduces memory hardware requirements.

- It replaces subroutines by individual words and related groups of words called Vocabularies. These are quickly modified and tested by editing 1024-character text blocks, called screens, using built-in editor.

tFORTH is a basic system implemented for SS-50 buss 6809 systems with the TSC FLEX 9.0 disk operating system. It is available on 5 1/4" or 8" single density soft-sectored floppy disks. **\$100.00**

tFORTH+ consists of tFORTH plus a complement of the following FORTH source code vocabularies: full assembler, cursor controlled screen editor, case statements, extended data types, general I/O drivers. **\$250.00**

firmFORTH is an applications package for use with tFORTH. It provides for recompilation of the tFORTH nucleus, deletion of superfluous code and production of fully rommable code. **\$350.00**

Call or write today.



3350 Walnut Bend, Houston, Texas 77042 • (713) 978-6933

machines, with little or no change. For example, it can be practical to download program development from a PDP-11 to a TRS-80 or an Apple II. It is even possible to write the software for a product before a hardware commitment is final.

Another advantage is that FORTH is a self-contained operating system. The 7 K bytes include terminal and disk handlers and a rudimentary file system. No other software is needed anywhere in the computer. Yet, if a monitor in read-only memory is available, FORTH can use it; and FORTH can run as a task under some other operating system (eg: CP/M) when that is wanted. FORTH can link together otherwise incompatible pieces of systems: software in read-only memory, operating systems, subroutine packages, and hardware. It provides a user interface that enables subroutine packages normally used by batch (ie: noninteractive) programs, mostly on older, larger computers, to be used interactively.

FORTH puts you in charge of your computer. You can understand everything happening in your soft-

ware or in any desired parts of it, and you can change it. This means no more "black box" systems that only the manufacturer's specialists can understand, no more dependence on someone else for upgrades, fixes, or documentation, and no more question of who is responsible if software does not work. The whole system is written in FORTH, right down to the bits—your application programs, the compiler, the operating system, the I/O drivers, etc. You do not have to learn some other language or be a systems specialist to modify it.

Disadvantages

Few FORTH systems used today have floating-point arithmetic. This is not a fault of the language; rather, it reflects its history in microcomputer control applications, where integer arithmetic is often needed for speed. Now there is more pressure for floating point, and it is becoming available.

A more fundamental limitation of FORTH is that it is not a typed language (unlike Pascal). For example, if an integer operation is per-

formed on a floating-point quantity, no message is printed either at compile time or at run time to warn of this error. (However, the user *can* add type checking and other error-preventing operations into any FORTH word.)

It may seem that unreliable code would result from the untyped nature of FORTH, but, in fact, FORTH code is remarkably solid and bug-free. The modularity and excellent testing environment aid error control; and type mismatches are less dangerous than most other mistakes because they are easy to detect.

Another criticism of FORTH is its lack of a directory file structure. Again, this is historical and is not a characteristic of the language, which can be developed to use any kind of files.

The traditional FORTH file system is primitive, but in practice it has worked very well. The entire disk (or disks) is a single virtual array of blocks numbered from 1, with the block size standardized at 1024 bytes regardless of physical disk sector size. The blocks (called *screens* because they can be displayed as sixteen 64-character lines on a terminal) are automatically buffered so that they are physically read and written only when necessary. A LOAD command will read a given screen and treat the information exactly as if it had been typed in a terminal session, thereby compiling source code or executing commands (depending on the contents of the screen). The LOAD instruction can be executed within a screen; in this way, a single LOAD command can control the compilation of large source programs.

This disk-based file system allows any part of the disk to be read or written with a single access. Load screens or data areas can be saved by name, and portions of the disk can be protected by redefining the names of a few input and output operations so that they check before writing and/or reading.

The disadvantage of this system is that there is no directory; when a new disk is inserted, the user or the program must know the block numbers for load screens and data files. Also, FORTH source programs are traditionally stored without tabs or truncation of blank lines, making white-space (ie: unused area on a line) and

ASCII encoded keyboards: Spillproof. Parallel or serial output, as low as \$69.*



RCA VP-600 series ASCII keyboards are available in two formats. You can choose either a 58-key typewriter format. Or a 74-key version which includes an additional 16-key calculator-type keypad. Both can be ordered with parallel or serial output.

These keyboards feature modern flexible membrane key switches with contact life rated at greater than 5 million operations. Plus two key rollover circuitry. A finger positioning overlay combined with light positive activation key pressure gives good operator "feel," and an on-board tone generator gives aural key press feedback.

The unitized keyboard surface is spillproof and dustproof. This plus high noise immunity CMOS circuitry makes these boards particularly suited for use in hostile environments.

Parallel output keyboards have 7-bit buffered, TTL compatible output. Serial output keyboards have RS 232C compatible, 20 mA current loop and TTL compatible asynchronous outputs with 6 selectable baud rates. All operate from 5 V DC, excluding implementation of RS 232C.

For more information contact RCA Customer Service, New Holland Avenue, Lancaster, PA 17604.

Or call our toll-free number: 800-233-0094.

*Optional user price for VP-601. Dealer and OEM pricing available.

CONDIMENTS

- ACCOUNTS PAYABLE - Tracks current and aged payables and incorporates a check writing feature. Maintains a complete vendor file with information on purchase orders and discount terms as well as active account status. Produces reports as follows: Open Voucher Report, Accounts Payable Aging Report and Cash Requirements. Provides input to PEACHTREE General Ledger. Supplied in source code for Micro-soft BASIC\$990/\$30
- ACCOUNTS RECEIVABLE - Generates invoice register and complete monthly statements. Tracks current and aged receivables. Maintains customer file including credit information and account status. The current status of any customer account is instantly available. Produces reports as follows: Aged Accounts Receivable, Invoice Register for Payment and Adjustment Register and Customer Account Status Report. Provides input to PEACHTREE General Ledger. Supplied in source code for Microsoft BASIC\$990/\$30
- PAYROLL - Prepares payroll for hourly, salaried and commissioned employees. Generates monthly, quarterly and annual reports. Provides employee W-2's. Includes tables for federal withholding and FICA as well as withholding for all 50 states plus up to 20 cities from pre-computed or user generated tables. Will print checks, Payroll Register, Monthly Summary and Unemployment Tax Report. Provides input to PEACHTREE General Ledger. Supplied in source code for Microsoft BASIC\$990/\$30
- INVENTORY - Maintains detailed information on each inventory item including part number, description, unit of measure, vendor and reorder data. Item activity and complete information on current item costs, pricing and sales. Produces reports as follows: Physical Inventory Worksheet, Inventory Price List, Departmental Summary Report, Inventory Status Report, The Reorder Point and the Period-to-Date and Year-to-Date reports. Supplied in source code for Microsoft BASIC\$1190/\$30

GRAHAM-DORIAN SOFTWARE SYSTEMS

- Comprehensive accounting software written in CBASIC-2 and supplied in source code. Each software package can be used as a stand-alone system or integrated with the General Ledger automatic posting to ledger accounts. Requires CBASIC-2.
- GENERAL LEDGER\$995/\$35
- ACCOUNTS PAYABLE\$995/\$35
- ACCOUNTS RECEIVABLE\$995/\$35
- INVENTORY SYSTEM\$590/\$35
- JOB COSTING\$995/\$35
- APARTMENT MANAGEMENT\$590/\$35
- CASH REGISTER\$590/\$35
- POSTMASTER - A comprehensive package for mail list maintenance that is completely menu driven. Features include keyed record extraction and label production. A form in menu mode and a status report provides neat letters on single sheet or continuous forms. Compatible with NAD files. Requires CBASIC-2\$150/\$15

STRUCTURED SYSTEMS GROUP

- GENERAL LEDGER - Interactive and flexible system providing proof and report outputs. Customization of COA created interactively. Multiple branch accounting centers. Extensive check and performed data entry for proof, COA correctness, etc. Journal entries may be batched prior to posting. Closing procedure automatically backs up input files. Now includes Statement of Changes in Financial Position. Requires CBASIC-2\$1250/\$25
- ACCOUNTS RECEIVABLE - Open item system with output for internal aged reports and customer-oriented statement and billing purposes. On-Line Enquiry permits information for Customer Service and Credit departments. Interface to General Ledger provided if both systems used. Requires CBASIC-2\$1250/\$25
- ACCOUNTS PAYABLE - Provides aged statements of accounts by vendor with check writing for selected invoices. Can be used alone or with General Ledger and/or with NAD. Requires CBASIC-2\$1250/\$25
- PAYROLL - Flexible payroll system handles weekly, bi-weekly, semi-monthly and monthly payroll periods. Tips, bonuses, reimbursements, advances, sick pay, vacation pay, and compensation time are all part of the payroll records. Prints government required periodic reports and will post to multiple SSIS General Ledger accounts. Requires CBASIC-2 and 54K of memory\$1250/\$25
- INVENTORY CONTROL SYSTEM - Performs controlling functions of adding and deleting stock items, adding new items and deleting old items. Tracks quantity of items on hand, on order and back-ordered. Optional hard copy audit trail is available. Reports include Master Item List, Stock Activity, Stock Valuation and Re-order List. Requires CBASIC-2\$1250/\$25
- ANALYST - Customized data entry and reporting system. User specifies up to 75 data items per record. Interactive data entry, retrieval, and update facility makes information management easy. Sophisticated report generator provides customized reports using selected records with multiple level break-points. For summarization, require check utility such as OSORT, SUPER-SORT or VSORT and CBASIC-2\$250/\$15
- LETTERIGHT - Program to create, edit and type letters or other documents. Has facilities to enter, display, delete and move text, with good video screen presentation. Design and create word NAD for form letter mailings. Requires CBASIC-2\$200/\$25
- NAD Name and Address selection system - Interactive mail list creation and maintenance program with output as full reports with reference data or restricted information for mail labels. Transfer system for extraction and transfer of records to create new files. Requires CBASIC-2\$100/\$20
- OSORT - Fast sort/merge program for files with fixed record length, variable field length information. Up to five ascending or descending keys. Full back-up of input files created\$100/\$20

- HEAD CLEANING DISKETTE - Cleans the drive Read/Write head in 30 seconds. Diskette absorbs loose oxide particles, fingerprints, and other foreign particles that might hinder the performance of the drive head. Lasts at least 3 months with daily use. Specify 5" or 8".
 - Single sided\$20 each/\$55 for 3
 - Double sided\$25 each/\$65 for 3
- FLIPPY DISK KIT - Template and instructions to modify single sided 5 1/4" diskettes for use of second side in single sided drives\$12.50
- FLOPPY SAVER - Protection for center holes of 5" and 8" floppy disks. Only 1 needed per diskette. Kit contains centering post, pressure tool and 14.95 7 mil mylar reinforcing rings for 25 diskettes.
 - 5" Kit\$14.95
 - 5", Rings only\$7.95
 - 8", Kit\$16.95
 - 8", Rings only\$8.95
- PASCAL USER MANUAL AND REPORT - By Jensen and Wirth. The standard textbook on the language. Recommended for use by Pascal/Z, Pascal/M and Pascal/MT users\$12
- THE C PROGRAMMING LANGUAGE - By Kernighan and Ritchie. The standard textbook on the language. Recommended for use by BDS C, Tiny C, and Whitesmiths C users\$12
- STRUCTURED MICROPROCESSOR PROGRAMMING - By the authors of SMALL80. Covers structured programming, the 8080/8085 instruction set and the SMALL/80 language\$20
- ACCOUNTS PAYABLE & ACCOUNTS RECEIVABLE - CBASIC - By Osborne/McGraw-Hill\$20
- GENERAL LEDGER - CBASIC - By Osborne/McGraw-Hill\$20
- LIFEBOT DISK COPYING SERVICE - Transfer data or programs from one media format to another at a moderate costfrom \$25

Hearty Appetite.

*CP/M and MP/M are trademarks of Digital Research. Z80 is a trademark of Zilog, Inc. UNIX is a trademark of Bell Laboratories. WHATSIT? is a trademark of Computer Headware. Electric Pencil is a trademark of Michael Shrayner Software. TRS-80 is a trademark of Tandy Corp. Pascal/M is a trademark of Sorcim.

1 Recommended system configuration consists of 48K CP/M, 2 full size disk drives, 24 x 80 CRT and 132 column printer.

2 Modified version available for use with CP/M as implemented on Heath and TRS-80 Model I computers.

3 User license agreement for this product must be signed and returned to Lifeboat Associates before shipment may be made.

4 This product includes/excludes the language manual recommended in Condidments.

5 Serial number of CP/M system must be supplied with orders.

6 Requires Z80 CPU.

Ordering Information

MEDIA FORMAT ORDERING CODES

When ordering, please specify format code.

LIFEBOT ASSOCIATES MEDIA FORMATS LIST

Diskette, cartridge disk and cartridge tape format codes to be specified when ordering software for listed computer or disk systems. All software products have specific requirements in terms of hardware or software support, such as MPU type, memory size, support operating system or language.

Computer system	Format Code
Atair 800 Disk See MITS 3200
AltosA1*
Apple + Microsoft SoftCardRG
BAF System 7100RD
Blackhawk Single DensityQ3
Blackhawk Microfills Mod IIQ2
CDS Versatile 3BQ1
CDS Versatile 4Q2
COMPAL-80Q2
Commoem System 3A1*
Commoem Z2DR6
CSBN BACKUP (tape)T1#
Digital Microterm IIRD
Digital MicrosystemsA1*
Discus See Morrow Discus
DurangoFR
Dynabyte DB2/2R1
Dynabyte DB2/4A1*
Exidy Sorcerer + Lifeboat CP/MQ4
Exidy Sorcerer + Exidy CP/MQ2
Health 88 + H17/H27P4
Health 88 + Lifeboat CP/MP4
Health 88 + Magnolia CP/MP7
Helios II See Processor Technology Helios II
Horizon See North Star ICOM 2411 Micro Floppy
ICOM 3712A1
ICOM 3812A1
ICOM 4511 5440 Cartridge CP/M 1.4 D1#D1#
ICOM 4511 5440 Cartridge CP/M 2.2 D2#D2#
IMS 6000RA
IMS 8000A1*
IMSAI VDP-40R4*
IMSAI VDP-42R5*
IMSAI VDP-44R5*
IMSAI VDP-80A1*
Intelec See ISC Intelec Interlist
Interlist MDS Single DensityA1
Interlist SuperBrain DOS 0.1R7
Interlist SuperBrain DOS 0.5-XR3
Interlist SuperBrain DOS 3.XRK
ISC Intelec 8063/8360/8963A1
Kornton P5i-80RF
Mecca 5 1/4"P6
Micromation (Except TRS-80 below)A1*
Microfills Mod IQ1
Microfills Mod IIQ2
MITS 3200/3202B1
Monitor DiscusA1*
MostekA1
MSD 5 1/4"RC
North Star Single DensityP1
North Star Double/QuadP2
Nytec Single DensityQ3
Nytec Microfills Mod. IIQ2
Ohio Scientific C3A3
Onyx 8200T2#
Perics PCC-2000B2
Processor Technology Helios IIB2
RAIR Single DensityR9

Computer system	Format Code
RAIR Double DensityRE
Research Machines 8"A1
Research Machines 5 1/4"RH
REX See Morrow Discus
SD Systems 8"A1*
SD Systems 5 1/4"R3
Sorcerer See Exidy Sorcerer
SpacebyteA1
SuperBrain See Interlist
TatbellA1*
TEI 8"R3
TEI 5 1/4"R3
TRS-80 Model I + FEC FreedomRN
TRS-80 Model I + MicromationA4*
TRS-80 Model I + Omikron 8"RM
TRS-80 Model I + Omikron 5 1/4"RM
TRS-80 Model I + Shuffboard 8"A1
TRS-80 Model IIA1*
VDP-40/42/44/80 See IMSAI
Vector M2Q2
Versatile See CDS Versatile
Vista V80 5 1/4" Single DensityP5
Vista V200 5 1/4" Double DensityP6
Zenith 288 + Lifeboat CP/MP4
Zenith 288 + Magnolia CP/MP7

*Single-Side Single-Density disks are supplied for use with Double-Density and Double-Side 8" soft sector format systems.

**IMSAI formats are single density with directory offset of zero.

#A media surcharge of \$25 for orders on tape formats T1 and T2 and of \$100 for orders on disk formats D1 and D2 will be added.

The list of available formats is subject to change without notice. In case of uncertainty, call to confirm the format code for any particular equipment.

space for comments costly in disk space and load time, discouraging good program layout. For these reasons, there is increasing interest in changing to a directory file system. Perhaps it will be written on top of the screen system currently in use.

The most important criticism of FORTH is that its source programs are difficult to read. Some of this impression results from unfamiliarity with a language different from others in common use. However, much of it results from its historical development in systems work and in read-only-memory-based machine control, where very tight programming that sacrifices clarity for memory economy can be justified. Today's trend is strongly toward adequate commenting and design for readability.

FORTH benefits most from a new, different programming style; techniques blindly carried over from other environments can produce cumbersome results. Most FORTH programmers seldom use named variables; they use the stack instead so that the implicit commenting normally available through choice of variable names is only provided through comments and user-defined operation names. Single definitions that would have more than about three unrelated numbers on the stack at any one time are best split into two or more operations; most programmers learn to keep their definitions short.

FORTH enforces extreme modularity, so the decomposition of each task into component parts is critical. Top-down design is especially important. Large jobs should be written as application-oriented libraries of operations to make teamwork and maintenance easier. A much larger fraction of the total programming effort is spent on design, with less on coding and debugging. For these and other reasons, FORTH creates its own issues of style, which are only beginning to be explored.

A Taste of FORTH

FORTH is an interactive language best explained by example. Because of this, a series of listings (listings 1 thru 10) with fairly detailed explanations make up the rest of this article. In the listings that follow, underlining denotes user keyboard input.

NEW! NEW! NEWSLETTER FROM LIFEBOT

- Latest Version Numbers List of Software
- Update on CP/M Users Group
- The Great ZOSO Speaks Out from Behind the Scenes

\$18 ppd. for 12 issues (U.S., Canada, Mexico). Elsewhere \$40. Send Check to "Lifelines," 1651 Third Avenue, New York, N.Y. 10028 or use your VISA or Mastercard - call (212) 722-1700

Lifeboat Associates
THE SOFTWARE SUPER-MARKET

Prices F.O.B. New York. Shipping, handling and C.O.D. charges extra. Manual cost applicable against price of subsequent software purchase. The sale of each proprietary software package conveys a license for use on one system only.

FORTH uses punctuation in some of its words, which makes representing them in text a difficult problem. For example, one FORTH word is ("), which could be taken to mean one of several character combinations. (For your information, the word has three characters and is made from a left parenthesis followed by a double quote mark and a right parenthesis.)

To decrease the chance of confusion while trying not to clutter text unnecessarily, we will sparingly use braces, {}, to isolate the character string within as a FORTH word or phrase. (For example, the above word would be written { (") } .) Braces will be used only under the following situations:

- when the material being quoted is a

phrase of FORTH words (eg: { 26 LOAD } or { 3 5 + })

- with the FORTH words { . } (period), { , } (comma), { : } (colon), { ; } (semicolon), { ? } (question mark), { ! } (exclamation point), { ' } (single quote mark), and { " } (double quote mark)
- with any word using the above punctuation marks (eg: { \$. } or { . " }).

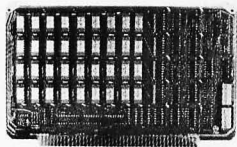
All other FORTH words will be set apart by a space on either side of the word. So, in this and other FORTH articles in this issue, braces will always signal a FORTH word or phrase. The braces are not part of the word or phrase, and FORTH words will never use braces within the body of a figure or listing....GW

On the Necessity of Using Camera-Ready Copy

Examination of listings 1 thru 10 will reveal a variety of typefaces used. This variety is present because each listing was created by the printer of the system producing the listing. Such listings are called camera-ready copy, which means that we can reproduce them in BYTE without inadvertently adding the errors that creep in with the retyping of a listing. Contributors to BYTE and onComputing are strongly encouraged to submit camera-ready listings made with a fresh ribbon, since this helps us to improve the accuracy of the article.

64KB RAM MEMORIES

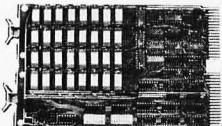
LSI-11 - \$750.00 ● SBC 80/10 - \$750.00
S-100 - \$750.00 ● 6800 - \$750.00 ● 6800-2 - \$995.00



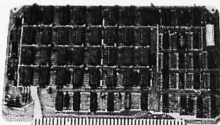
CI-6800-2 64K x 9



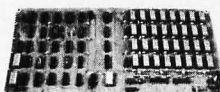
CI-S100 64K x 8



CI-1103 32K x 16



CI-6800 64K x 8



CI-8080 64K x 8

CI-6800-2 — 16KB to 64KB. Plugs directly into Motorola's EXORciser I or II. Hidden refresh up to 1.5 Mhz. Cycle stealing at 2 Mhz. Addressable in 4K increments with respect to VXA or VUA. Optional on Board Parity. 64K x 9 \$995.00.

CI-S100 — 16KB to 64KB. Transparent hidden refresh. No wait states at 4 Mhz. Compatible with Alpha Micro and all Major 8080, 8085 and Z80 Based S100 Systems. Expandable to 512 K bytes thru Bank Selecting. 64K x 8 \$750.00.

CI-1103 — 16KB to 64KB on a single dual height board. On board hidden refresh. Plugs directly into LSI 11/2, H11 or LSI 11/23. Addressable in 2K word increments up to 256 K Bytes. 8K x 16 \$390.00. 32K x 16 \$750.00.

CI-6800 — 16KB to 64KB on a single board. On board hidden refresh. Plugs directly into EXORciser I and compatible with Rockwell's System 65. Addressable in 4K increments up to 64K. 16K x 8 \$390.00. 64K x 8 \$750.00.

CI-8080 — 16KB to 64KB on a single board. Plugs directly into MDS 800 and SBC 80/10. Addressable in 4K increments up to 64K. 16 KB \$390.00. 64K \$750.00.

Test and burned-in. Full year warranty.



Chrislin Industries, Inc.

Computer Products Division

31352 Via Colinas • Westlake Village, CA 91361 • 213-991-2254

Listing 1: FORTH as a calculator. FORTH is easy to approach because it can be used as a calculator. Here, the programmer has not defined any new operation but has used addition, multiplication, and print (the dot means print). These are three of about one hundred operations that are available when FORTH first comes up. Programming consists of defining new operations which can be custom designed for a particular task or a particular industry.

FORTH uses postfix (also called RPN or reverse Polish notation) arithmetic, which is best known from its use in Hewlett-Packard calculators. In postfix notation, the operations are written after their arguments, not between them. The text of this article shows how postfix notation works, using a data structure called the stack, and it explains the formulas in this example.

Postfix notation, which does not use parentheses, is more general than ordinary arithmetic notation. Its biggest advantage is that it greatly simplifies the writing and calling of subroutines.

In these examples, underlining indicates what the user has typed on the terminal. FORTH does not process the line until you type a carriage return. The OK prompt means that the system has completed its work and is ready for new input from the user.

```

2 3 + . 5 OK
4 5 + 6 7 + * . 117 OK

```

Listing 2: Changing number bases. FORTH can work in different number bases and can change any time, so it serves as an octal/hexadecimal/binary/decimal calculator within the limits of 16-bit numbers (or 32 bits for double precision). The FORTH word HEX converts FORTH into a hexadecimal machine, and all numbers are printed in

Listing 2 continued on page 112

ELCOMP Books

Care and Feeding of the Commodore PET

Eight chapters exploring PET hardware. Includes repair and interfacing information. Programming tricks and schematics.
Order No. 150 \$11.00

8K Microsoft BASIC Reference Manual

Authoritative reference manual for the original Microsoft 4K and 8K BASIC developed for Altair and later computers including PET, TRS-80, and OSI. OSI owners please take note!
Order No. 151 \$9.95

Expansion Handbook for 6502 and 6802

(S-44 Card Manual) Describes all of the 4.5 x 6.5 44 pin S-44 cards incl. RAM, ROM, dig. I/O, MUX/A to D, EPROM Prog. etc. With schematics and funct. descriptions. A must for every KIM, SYM and AIM owner.
Order No. 152 \$9.95

Microcomputer Application Notes

Reprint of Intel's most important application notes, including 2708, 8085, 8255, 8251 chips. Very necessary for the hardware buff.
Order No. 153 \$9.95

Complex Sound Generation

New, revised applications manual for the Texas Instruments SN 76477 Complex Sound Generator. Circuit board available (\$8.95)
Order No. 154 \$6.95

Small Business Programs

Complete programs for the business user. Mailing List, Inventory, Invoice Writing and much more. Introduction into Business Applications. Many listings.
Order No. 156 \$14.90

The First Book of Ohio Scientific, Vol. I

Contains an introduction to personal computers and describes the Ohio Scientific Line. Contains explanatory diagrams: block, hook-up, expansion, tricks, hints and many interesting listings. Hardware and software information not previously available in one compact source. 192 pages.
Order No. 157 \$7.95

The First Book of Ohio Scientific, Vol. II

Vol. II contains very valuable information about Ohio Scientific microcomputer systems. Introduction to OS-650 and OS65-U, networking and distributed processing, systems specifications, business applications, hard and software hints and tips.
Order No. 158 \$7.95

Mailing List Program for Challenger C1/C2 8K

Order No. 2004 - Personal Version \$9.95

Order No. 2005 - Business Version \$9.95

Ohio Scientific Expansion Information

Conversion of CIP (Cassette) to 52x26 display. Detailed step by step instructions for doubling the CIP speed and display size!
Order No. 1105 \$12.00

Important Software for CBM 16K/32K

Most powerful Editor/Assembler for Commodore CBM 16/32K on cassette. Very fast—Editor divides screen into 3 parts. Scrolling text window, 24 direct commands, 19 serial commands, status and error messages. Assembler can be started directly from the editor or from the TIM monitor. Translates in three passes. If an error is encountered, automatic return to the editor. Cassette with DEMO.
Order No. 3276 \$69.00

ATTENTION APPLE USERS

Same as above for Apple II or Apple II plus.
Order No. 3500 \$89.00

MONJANA/1 makes Machine Language Programming easy!

In every Commodore CBM there is a spare ROM socket waiting for its MONJANA/1. The new MONJANA/1 Machine Language Monitor in ROM offers more user guidance and debugging aids than any other monitor available today. It is indispensable for anyone intending to take full advantage of the computers features. Trace, link, disassemble, dump, relocate, line assemble and much more. Every command function has demand printout option. Price includes extensive manual.
Order No. 2001 \$98.00

JANA-Monitor on cassette for the PET

Similar to MONJANA/1 - very powerful.
Order No. 2002 \$29.00

ELCOMP PUBLISHING Inc.

3873-L Schaefer Ave., Chino CA 91710 (714) 591-3130

Please send me the books/software indicated below

I enclose \$_____ send postpaid

Send COD (\$5 extra)

Charge my VISA Mastercharge

Acct. No. _____

Exp. date _____ Signature _____

Book No. _____ Book No. _____ Software No. _____

1 Year subscription to ELCOMP Newsletter \$9.80

Name _____ Phone _____

Address _____

City _____ State _____ Zip _____

CA add 6% sales tax. We also accept Eurocheck. All orders outside USA must add 15% shipping.

Listing 2 continued:

hexadecimal until some other operation changes the base again. FORTH always begins a session in decimal radix.

The operations DECIMAL and HEX are built into the system; OCTAL, BINARY, and TRINARY (base 3) are not. So when OCTAL was first used, the error message { OCTAL ? } indicated an undefined word; that is, the system did not recognize the word OCTAL. In the next line, the user defined OCTAL (line 6). This example illustrates FORTH's extensibility; users can extend the language to include new operators.

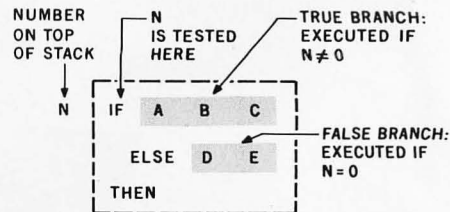
Incidentally, the second error message { 12885 ? } in line 12 resulted because the system was in binary (from the line above), and, in binary, numbers must contain only the digits 0 and 1, so 12885 was not recognized as a number. It was treated as a word, and, because there was no operation named 12885, the error message was generated.

OCTAL and the other number-base operations work by giving a new value to BASE, a variable used by the system. Defining new operations is more fully explained in listing 3. The { ! } operation (store) is explained later.

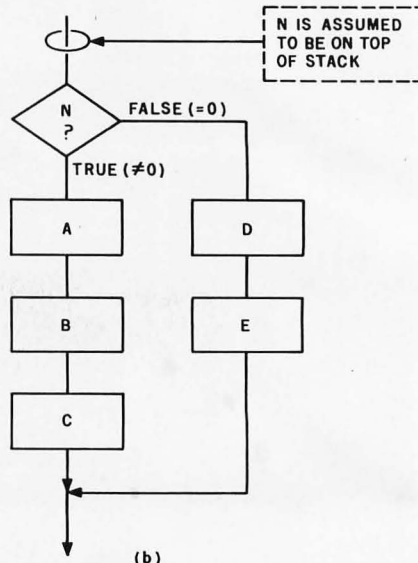
Number bases only affect input and output. All internal computation is in binary, so there is no speed penalty for using nondecimal numeric bases.

```

HEX OK
3BE8 C8 + . 3CBO OK
25 2F * . 6CB OK
DECIMAL 1348 HEX . 544 OK
DECIMAL 1348 OCTAL . OCTAL ?
: OCTAL 8 BASE ! ; OK
DECIMAL 1348 OCTAL . 2504 OK
DECIMAL OK
: BINARY 2 BASE ! ; OK
: TRINARY 3 BASE ! ; OK
12885 BINARY . 11001001010101 OK
12885 BINARY . 12885 ?
DECIMAL 12885 BINARY . 11001001010101 OK
DECIMAL 12885 OCTAL . 31125 OK
DECIMAL 12885 HEX . 3255 OK
DECIMAL 12885 TRINARY . 122200020 OK
DECIMAL -12885 TRINARY . -122200020 OK
DECIMAL OK
    
```



(a)



(b)

Figure 2: An explanation of the IF...ELSE...THEN construct. (See listing 4.) As shown in figure 2a, the portion of code executed depends on the value of the number on top of the stack when the word IF is encountered. If we call this number N and say that the number has a boolean value of true if its numeric value is nonzero and false if 0, then figure 2b gives the equivalent construct to figure 2a in conventional flowchart notation. Here and in figures 3 thru 5, the dotted box indicates the boundaries of the construct (as opposed to values assumed to be on the stack).

Listing 3: Defining new operations. Here, a new operation CUBE is created. CUBE replaces whatever number is on top of the stack with the cube of that number. The statements within the parentheses are comments.

The colon, { : }, begins a FORTH word definition; the word following it is the name being defined. Semicolon, { ; }, ends the definition.

The new word CUBE will first execute DUP, which duplicates the number on top of the stack, making a second copy. The second DUP leaves three copies. The first * causes the top two copies to be replaced by the square of the number; the next * computes the cube, and then all three copies of the original number are gone, leaving the cube of the number on top of the stack.

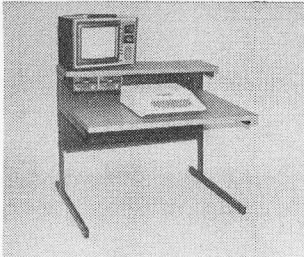
This colon definition shows one of several ways to create new words in FORTH. Most words that appear inside the definition are compiled and not executed immediately.

All words and numbers in FORTH are separated by one or more blanks (and/or carriage returns). FORTH operation names can be up to thirty-one characters long and can consist of letters, numbers, or any other characters. For example, an operation name could be a number, or it could be nonprinting characters only. In practice such names are rarely used, but they illustrate the flexibility that is available.

Listing 3 continued on page 114

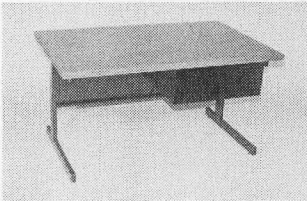
CHOOSE...

Choose an Apple Desk



A compact bi-level desk ideal for an Apple computer system. This 42" x 31 1/2" desk comes with a shelf to hold two Apple disk drives. The top shelf for your TV or monitor and manuals can also have an optional paper slot to accommodate a printer.

Choose a Micro Desk



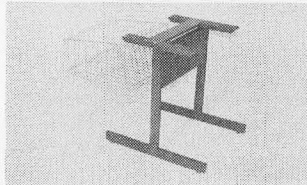
Get your micro computer off the desk top and into the micro shelf under our Designer Series desks. Suitable for the North Star, Dynabyte, Vector Graphics, and Altos computers. The desks come in a variety of sizes and colors.

Choose a Mini Rack



Mini racks and mini micro racks have standard venting, cable cut outs and adjustable RETMA rails. Choose a stand alone bay or a 48", 60", or 72" desk model in a variety of colors and wood tones. A custom rack is available for the Cromemco.

Choose a Printer Stand



The Universal printer stand fits the:

- Centronics 700's
- Diablo 1600's & 2300's
- Dec LA 34
- T.I. 810 & 820
- NEC Spinwriter
- Okidata Slimline
- Lear Siegler 300's
- Anadex 9500's

Delivery in days on over 200 styles and colors in stock. Dealer inquiries invited.

ELECTRONIC SYSTEMS FURNITURE COMPANY

17129 S. Kingsview Avenue
Carson, California 90746
Telephone: (213)538-9601

Listing 3 continued:

This listing shows CUBE being executed from the terminal. It can also be used as a component in further definitions. A fundamental property of FORTH is that operations defined by users are indistinguishable from those which were originally part of the system.

```
: CUBE ( N -> N. CUBE A NUMBER)
  DUP DUP ( NOW THERE ARE THREE COPIES)
  * * ( GET THE CUBE)
  ; OK
5 CUBE . 125 OK
-28 CUBE . -21952 OK
HEX 17 CUBE BINARY . DECIMAL 10111110000111 OK
```

Listing 4: Conditional branching. The IF ... THEN is for conditional execution. IF takes one argument off of the stack; this argument is interpreted as a boolean or truth value, with 0 meaning false and any nonzero value meaning true. If true, any statements between the IF and THEN are executed. In either case, execution continues after the THEN, which terminates the conditional. There is also an optional ELSE clause that is executed only if the argument is false. (See figure 2.)

Here, the true-clause contains only one word, MINUS, but it could contain almost any FORTH statements, including other conditionals and loops nested to any practical depth. These statements run fast because they are compiled into a form of object program called threaded code.

Incidentally, the FORTH word 0< returns a boolean value indicating whether its argument (the number on top of the stack) is less than zero. The DUP is necessary because 0< follows the FORTH convention that operations should destroy their arguments on the stack. MINUS reverses the sign of its argument (the top stack number).

Items in parentheses are comments. The comment "N -> N" in the first line is to show that this operation takes one number off of the stack and returns one number to it. Perhaps the most important information to put in the comments accompanying each new operation is what arguments it takes off of the stack and what results it returns to the stack.

```
: ABSOLUTE-VALUE ( N -> N. ABSOLUTE VALUE)
  DUP 0< ( GET BOOLEAN, TRUE IF NEGATIVE)
  IF MINUS THEN ( NEGATE THE NUMBER IF TRUE)
  ; OK
10 ABSOLUTE-VALUE . 10 OK
-5 ABSOLUTE-VALUE . 5 OK
```

Listing 5: The DO ... LOOP, a structured loop with a counting index. DO takes two arguments from the stack, the initial value of the index (on top) and the final value plus 1. (See figure 3.) These indices are written in reverse order from most other languages, making the loop terminating value (which is more often passed as an argument) more accessible on the stack.

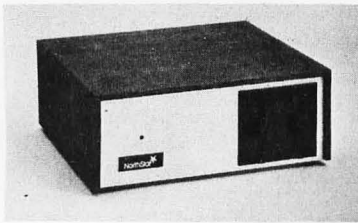
CR simply performs a carriage return. In this example, the index values are literals (10 and 0), but they can also come from variables or from computations of any complexity; anything that gets the indices onto the stack is legitimate.

This listing also shows a timing benchmark; the word TIME-TEST does 30,000 empty loops. On an Apple II running FORTH, TIME-TEST executes in less than 4 seconds. In Apple Integer BASIC (which is a fast BASIC), 30,000 empty loops take 40 seconds.

```
: 10CUBES ( ->. PRINT A TABLE OF CUBES OF 0-9)
  10 0 ( INDICES OF LOOP)
  DO ( START LOOP)
    CR I . I CUBE . ( PRINT A NUMBER AND ITS CUBE)
  LOOP ( END OF LOOP)
  ; OK
10CUBES
0 0
1 1
2 8
3 27
4 64
5 125
6 216
7 343
8 512
9 729 OK
: TIME-TEST 30000 0 DO LOOP ; OK
TIME-TEST OK
```

SAVE MORE THAN 20%
NORTH STAR—INTERTUBE—MICROTEK
ZENITH—HEATH—ITHACA
THINKER TOYS—GODBOUT—SOFTWARE

The smartest computers at the smartest price



FACTORY ASSEMBLED & TESTED	LIST	ONLY
HORIZON-1-16K-DOUBLE DEN KIT	SPECIAL	\$1269
HORIZON-1-32K-DOUBLE DEN KIT	\$1999	1575
HORIZON-2-32K-DOUBLE DEN KIT	2399	1879
HORIZON-1-32K-DOUBLE DEN	2695	2129
HORIZON-2-32K-DOUBLE DEN	3095	2435
HORIZON-2-32K-QUAD DENSITY	3595	2839
HORIZON-2-64K-QUAD+HARD DISK	9329	7229
HORIZON MEMORY 16K 389	32K	579
NORTH STAR HARD DISK 18 Mb	4999	3949
PASCAL FOR NORTH STAR ON DISK	199	190
Powerful NORTH STAR BASIC..The Best.....	FREE	
2 NORTH STAR SOFTWARE DISKS w/HORIZON.....	FREE	
NORTH STAR BUSINESS PROGRAMS & NORTHWORD.	PHONE	
COLOR! RAINBOW-2000 & CAT-100	PHONE	
ITHACA FRONT PANEL COMPUTER 64K	2885	2449
Z-8000 CPU CARD 16-bit ITHACA S-100	PHONE	
ITHACA MEMORY 8/16-bit	PHONE	
8086 CPU 16 bit 10xfaster SEATTLE COMPUTER	PHONE	
SEATTLE COMPUTER MEMORY	PHONE	
SSM Z-80 CPU, VIDEO BOARD, MEMORY	PHONE	
MEASUREMENT MEMORY 64K A & T 4mHz	650	
JAWS MEMORY 64K A & T 4mHz	PHONE	
GODBOUT MEMORY - Static, Super Selection & Price	PHONE	
THINKER TOYS DISCUS/2D A & T	1199	975
THINKER TOYS HARD DISK 26 Mb	4995	4149
DISCUS/2+2 1.2 Mbytes A & T	1545	1285
THINKER TOYS SUPERRAM	PHONE	
DELTA COMPUTER & DISK DRIVES	PHONE	
TARBELL COMPUTERS & DISK DRIVES	PHONE	
INTERTUBE II SMART TERMINAL	995	725
ZENITH-HEATH SMART TERMINAL Z-19 A & T	795	



ZENITH COMPUTER-TERMINAL-DISK Z-89	2595	2195
CAT NOVAION MODEM	179	169
MICROTEK PRINTER	795	725
AXIOM PRINTER	795	695
ANADIX PRINTER	995	865
NEC PRINTER Fast Typewriter Quality	2915	2799
SECRETARY WORD PROCESSOR The Best!	85	77
TEXTWRITER III Book Writing Program	125	112
GOFAST NORTH STAR BASIC Speeder Upper	79	71
PDS Super Z-80 ASSEMBLER & More	99	89
COMPILER FOR NORTH STAR \$150 w/PDS & HDS	90	90
EZ-80 MACHINE LANGUAGE TUTORIAL \$25	HDS	40
EZ-CODER Translates English to BASIC	79	71
ECOSOFT FULL ACCOUNTING PKG	350	315
DATABASE, THE SOURCE 90, CROSS ASSEMBLERS-CALL	30	
BOX OF DISKETTES 29 IN PLASTIC CASE	30	
Which Computers are BEST? BROCHURE.....	FREE	
North Star Documentation refundable w/HRZ	20	
ORDER 2 or more COMPUTERS.... BIGGER DISCOUNTS	FACTORY ASSEMBLED & FACTORY WARRANTY	

AMERICAN
SQUARE COMPUTERS
 KIVETT DR * JAMESTOWN NC 27282
 (919)-889-4577

Listing 6: The BEGIN ... UNTIL loop. This loop takes one argument, a truth value, usually computed within the loop, at the end. If it is false (0), control branches back to the corresponding BEGIN ; if the value is true (nonzero), the loop ends, and control transfers to the next word in the program. (See figure 4.)

Note that the test of the value on top of the stack occurs at the end of the body of the loop; this guarantees that the body of the loop will be executed at least once.

The word = removes the top two numbers from the stack and returns a truth value of 1 if they are equal, 0 otherwise. In this example, the index stays on the stack and is duplicated before each use. The DROP at the end throws away the top stack value; this prevents the used index from cluttering the stack.

The warning message "10CUBES ISN'T UNIQUE" notifies us that the same name has already been defined. The only penalty for reusing a name is that the former definition becomes inaccessible for the rest of the program. Therefore, you do not have to remember a list of reserved words in FORTH; if you do not know about a name or have forgotten about it, you probably were not planning to use it anyway. But, in case of a mistake, the bad definition can be deleted with a FORGET operation, or the source code can be changed on disk.

[Some versions of FORTH use BEGIN ... END instead of BEGIN ... UNTIL GW]

```
: 10CUBES ( -> . SAME, USING 'UNTIL' LOOP) 10CUBES ISN'T UNIQUE
0 ( INITIAL VALUE OF INDEX)
BEGIN ( START LOOP)
  CR DUP . DUP CUBE . ( PRINT A # AND ITS CUBE)
  1 + ( INCREMENT)
  DUP 10 = ( TEST FOR INDEX=10)
UNTIL ( END OF LOOP)
DROP ( THROW AWAY USED INDEX)
: OK
10CUBES
0 0
1 1
2 8
3 27
4 64
5 125
6 216
7 343
8 512
9 729 OK
```

Listing 7: The BEGIN ... WHILE ... REPEAT loop. This looping structure tests the value on top of the stack at the beginning of the loop; because of this, this loop can execute 0 times. REPEAT causes an unconditional branch back to BEGIN , and WHILE branches out of the loop (just beyond REPEAT) if the truth-value which it finds on top of the stack is false (ie: 0); see figure 5.

All of these looping and conditional branching structures can be nested within each other to any practical depth. Any mismatching can be detected at compile time. Most FORTH systems allow these structures only inside colon definitions; they cannot be executed directly from the terminal.

[Some versions of FORTH use: BEGIN ... IF ... WHILE or WHILE ... PERFORM ... PEND instead of BEGIN ... WHILE ... REPEAT GW]

```
: 10CUBES ( -> . SAME, USING 'WHILE' LOOP) 10CUBES ISN'T UNIQUE
0 ( INITIAL VALUE OF INDEX)
BEGIN
  DUP 10 < ( LOOP TEST)
  WHILE
    CR DUP . DUP CUBE . ( PRINT A # AND ITS CUBE)
    1 + ( INCREMENT)
  REPEAT
  DROP ( THROW AWAY USED INDEX)
: OK
10CUBES
0 0
1 1
2 8
3 27
4 64
5 125
6 216
7 343
8 512
9 729 OK
```

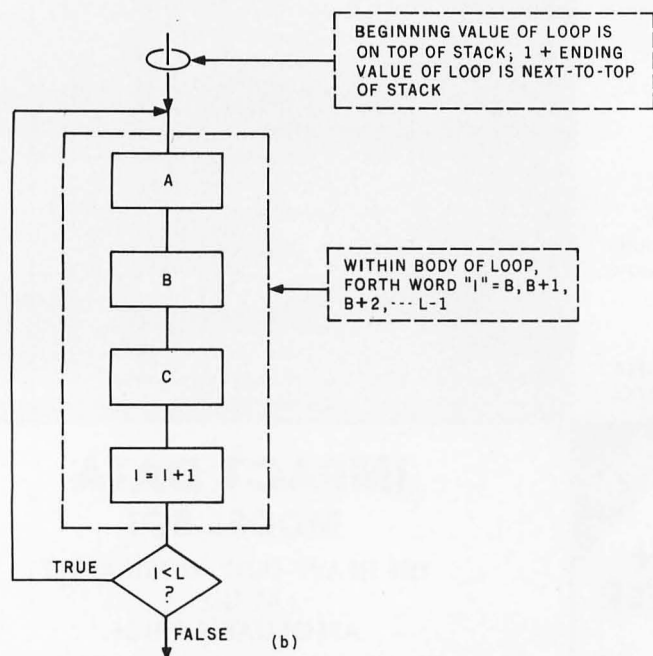
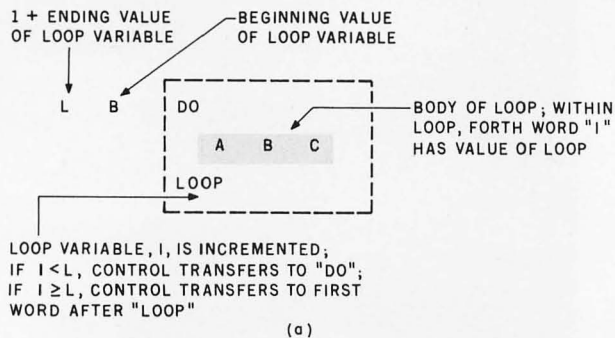



Figure 3: An explanation of the DO ... LOOP construct. As shown in figure 3a, the top number on the stack is taken to be the lower limit of the loop variable, L, and the next-to-top number on the stack is the upper limit of the loop variable + 1. The body of the loop is shaded, and the loop variable is incremented and tested after the body of the loop is executed. Figure 3b gives the equivalent construct in conventional flowchart notation.

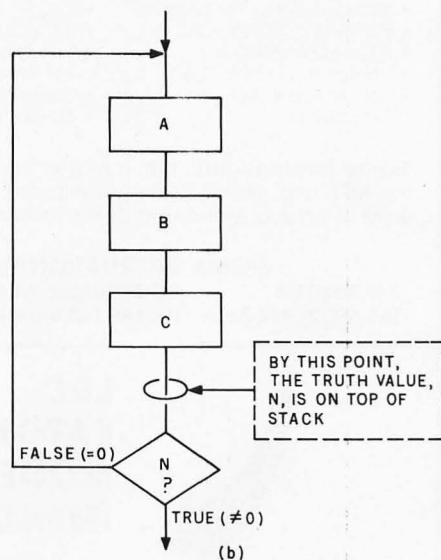
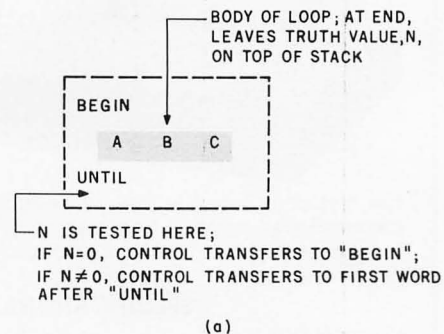


Figure 4: An explanation of the BEGIN ... UNTIL construct. As shown in figure 4a, the body of the loop (shaded) is repeated only if the value on top of the stack when the word UNTIL is reached is false. Figure 4b gives the equivalent construct in conventional flowchart notation.

FMG CORPORATION

5280 Trail Lake Drive Suite 14 Ft. Worth, Texas 76133 (817) 294-2510

CP/M is a registered trademark of Digital Research Corp.
TRS-80 is a registered trademark of Radio Shack.

FROM THE ORIGINATOR OF THE TRS-80 PROJECT

FMG Corporation — for HIGH LEVEL LANGUAGES

- FORTRAN • BASIC
- PASCAL • COBOL

Microcomputer software for business applications, engineers, consumers, hobbyists and others who have a serious interest in computers.



SEND FOR FREE SOFTWARE CATALOG

- CP/M — Industry Standard Operating System
- USCD PASCAL PACKAGE
- GENERAL LEDGER; PAYROLL; ACCOUNTS RECEIVABLE and ACCOUNTS PAYABLE
- FORTRAN-80 PACKAGE — New Capabilities for TRS-80 Users
- FMG's MICRO COBOL — For TRS-80 and TRS-80 Model II
- CP/M Z80 — Macro Assembler
- ZSID — Symbolic Debugger
- Custom Programming, Service, Installation and Training are Available at Additional Cost

FMG Corporation is an Independent Software Company — from the ORIGINATOR OF THE TRS-80 PROJECT and THE AUTHOR OF THE FIRST CP/M FOR THE TRS-80.

M-452

UNBEATABLE...



APPLE II OR APPLE II PLUS



Shipped direct to you!

\$899⁰⁰

(Plus Shipping)

We have orchard fresh Apple products ready to ship. Immediate delivery. Send cash or cashiers check for quick shipment. Orders with personal checks shipped after bank clearance.

- 16K UNITS.....\$899
32K UNITS.....\$999
48K UNITS.....\$1099
Apple Disk Drive \$550
Pascal Language Card \$450

Above plus \$20 shipping charge. IMPORTANT—No shipments made within the state of Illinois.

MIGHTY MICROS P.O. BOX 11375 CHICAGO, IL 60611

ORDER FORM

Enclosed \$

For Via U.P.S.

Ship to:

Name

Address (No P.O. Boxes—Street Address Only)

City

State Zip

Listing 8: An example of FORTH looping. A practical use of FORTH's structured looping is this terminal output handler. This example is for a PDP-11; an example for other computers would be similar. Address 177564 (octal) is the output status register of the console terminal; bit 7 of this address is set when the device is ready to receive a character. The ASCII code for the output character can then be placed in address 177566 (the data buffer register).

The FORTH word @ (pronounced fetch) does the work of PEEK in BASIC; it treats the number on top of the stack as an address and replaces it with the contents of that address word. AND does a "bitwise" boolean AND operation. So { 177564 @ 200 AND } indicates true (nonzero) only if bit 7 of the status register is set. Until then, the BEGIN ... UNTIL loop does a waiting loop ending on the above condition. When the device is ready, the argument that was given to TERMINAL-OUT (the ASCII character to be written) is still on top of the stack. { ! } (pronounced store) stores the word that is second on the stack into the address that is on top of the stack; so { 177566 ! } transmits the character to the terminal data buffer register, from which it will be written onto the terminal by the hardware of the PDP-11 system.

The FORTH word ASCII-TEST was written to test the TERMINAL-OUT word. It transmits ASCII values for all of the printable character set.

Listing 9 shows the same device handler, only written in machine-language code with a FORTH assembler.

```
OK
OCTAL OK
: TERMINAL-OUT ( CHAR -> , TERMINAL OUTPUT HANDLER, PDP-11)
  BEGIN 177564 @ 200 AND UNTIL ( WAIT TILL PORT READY)
  177566 ! ( TRANSMIT THE CHARACTER)
: OK
: ASCII-TEST ( - ) , TEST HANDLER - PRINT CHARACTER SET)
  177 40 ( TRANSMIT ASCII BLANK THROUGH '+' )
  DO I TERMINAL-OUT LOOP ( OUTPUT THE CHARACTERS)
: OK
DECIMAL OK
ASCII-TEST !"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN0PQRSTUUVWXYZ[\]^_`abcd
efghijklmnopqrstuvwxyz{|}~* OK
```

Listing 9: FORTH words defined by machine-language subroutines, for PDP-11 and for 8080 processors. The operation TERMINAL-OUT-2 behaves exactly the same as TERMINAL-OUT defined in listing 8, but it is written in assembly language. FORTH assemblers use postfix notation, so address-mode symbols and operation codes (instruction mnemonics) follow their operands, unlike conventional assemblers. In the PDP-11 example (listing 9a), { 177564 200 # BIT, } in line 2 assembles a "bit test" instruction that does a logical AND between address 177564 and the literal 200 (# indicates literal), setting condition codes. { UNTIL, } assembles a conditional branch back to the corresponding { BEGIN, }. The commas are part of the operation names, not punctuation. The word NE tells the { UNTIL, } what kind of conditional branch to assemble. There are also { IF, }...{ THEN, } and { IF, }...{ ELSE, }...{ THEN, } operations; all these code-level structures can be nested.

In the 8080 example (listing 9b), the machine-language subroutine sets up a call to the character-output routine in the North Star disk operating system. In contrast, the PDP-11 example outputs directly to the hardware without using any software outside of FORTH. Either approach could be used on either machine, of course, and each has its own advantages.

The word CODE, like { : } (colon, introduced in listing 3), creates a new definition in FORTH's dictionary for the word following it. CODE also sets the number base (to octal for PDP-11 and to hexadecimal for 8080), saving the original number base, which is later restored by { C; }. CODE also changes the vocabulary, which allows the same names to have different meanings in the assembler and in the rest of FORTH without confusion. Users can create their own vocabularies and subvocabularies to keep different application libraries separate.

Many FORTH programmers never need to write machine-language subroutines, so they do not need to use an assembler. FORTH assemblers have an unfamiliar postfix notation, but they have the advantage of giving immediate feedback. You know right away whether an operation works, with no wait for assembly passes, linking passes, and file handling. This interactive assembly greatly speeds program development and allows more thorough testing.

Listing 9 continued on page 122

CP/M® SOFTWARE

8080 Emulator

RAID is a software-based system rivaling hardware emulators costing thousands of dollars. RAID is absolutely the most advanced and sophisticated debugging system ever developed for a computer. Fully symbolic, including labels, operands and op-code mnemonics, RAID combines real-time and emulation modes in a single package. Tracing by *prime path*, *individual instructions*, *subroutines* and *breakpoints* is supported. Special feature allows emulation and real-time modes to function together for high speed emulations. Other features include memory search facilities, disk access by track and sector, single-step, multi-step, block move, user-selectable radix, etc. Over 70 commands in all. Requires 24K min. CP/M^{®2} system.

Raid\$195
Manual only\$ 25

ISIS¹ Conversion

ISIS¹ to CP/M[®] conversion utilities permit CP/M[®] users to read or write files to or from an ISIS¹ diskette. The package consists of three utility programs that *read*, *write* and display the ISIS¹ *directory*.

ISIS¹ - CP/M[®] Utilities\$160
Manual only\$ 5

Floating Point Package

'FPP' is a set of 8080 assembly language subroutines that provide 12 digit BCD arithmetic functions for *add*, *subtract*, *multiply*, and *divide*. BCD arithmetic means no conversion errors and minimal conversion time. Source code is supplied on standard 8" diskette.

FPP on CP/M[®] diskette\$200
FPP on ISIS¹ diskette\$200
Manual only\$ 10

¹ISIS is a trademark of Intel Corporation.
²CP/M[®] is a registered trademark of Digital Research.



586 Shades Crest Road
Birmingham, Al.

Send check or money order to:
P.O. Box 3373 A
Birmingham, Al. 35205
Phone: 205 933-1659

Listing 9 continued:

Collectively, { : } and CODE are called defining words because they are used to create new FORTH words. There are several other such functions in FORTH, and users can also define their own types of defining words, creating new data types or operation types; see listing 10.

```

CODE TERMINAL-OUT-2 ( CHAR -> , TERMINAL OUTPUT HANDLER, PDP-11) OK
  BEGIN, 177564 200 # BIT, NE UNTIL, ( WAIT TILL PORT READY) OK
  $ )+ 177566 MOV, ( POP FORTH STACK INTO DATA REGISTER) OK
  NEXT, ( A 2-INSTRUCTION MACRO TO CONTINUE FORTH EXECUTION) OK
  C; ( GET OUT OF THE FORTH ASSEMBLER) OK
OCTAL OK
: ASCII-TEST-2 ( -> , PRINT ASCII CHARACTER SET)
  177 40 ( ASCII BLANK THROUGH ' ')
  DO I TERMINAL-OUT-2 LOOP ( OUTPUT THE CHARACTERS)
  ; OK
DECIMAL OK
ASCII-TEST-2 !"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN0PQRSTUVWXYZ[\]^_`ab
cdefghijklmnopqrstuvwxyz{|}~ OK
ASCII-TEST !"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN0PQRSTUVWXYZ[\]^_`abcd
efghijklmnopqrstuvwxyz{|}~ OK

0 CONSTANT DEV ( DEVICE NO FOR NORTHSTAR DOS ) OK
200D CONSTANT COUT ( NORTHSTAR DOS CHAR OUT JUMP POINT ) OK
CODE TERMINAL-OUT-2 ( CHAR->. 8080 WITH NORTHSTAR DOS ) OK
  H POP ( CHARACTER IS ON STACK, POP TO HL ) OK
  B PUSH ( BC IS INSTRUCTION POINTER, SAVE IT ) OK
  L B MOV ( DOS EXPECTS CHAR IN B REGISTER ) OK
  DEV A MVI ( AND DEVICE NUMBER IN ACCUMULATOR ) OK
  COUT CALL B POP NEXT JMP C; ( DO IT AND CONTINUE ) OK

```

Listing 10: User-defined data types. Because this example is longer, it was not typed in directly like the others, but was stored on disk with an editor (the editor session is not shown here). This example is contained in two disk screens, each of which is a virtual block of 1024 bytes (see text). The commands { 58 LIST } and { 59 LIST } print these screens. The line numbers (0 thru 15) are not part of the program and are used only by the editor.

This example creates table-lookup sine and cosine routines for integer-degree arguments. The results are accurate enough for most graphics applications, making this situation an example of the versatility of FORTH, even without floating-point routines.

The definition of TABLE creates a new data type. When TABLE is executed, it creates a new table of numbers taken from the stack; the number on top of the stack tells how many items there are in the table. In this case, { 91 TABLE SINTABLE } creates a table called SINTABLE with ninety-one entries; these entries are the values of the sine of 0° thru 90°, multiplied by 10,000 so that they can be expressed as integers. SINTABLE gives the sine (scaled by 10,000) of 0° thru 90° degrees; SIN does the same, except that its argument can be any number of degrees (from -32,768 to 32,767).

Incidentally, few FORTH programs use as much depth of stack as this one. The system used for listings 1 thru 7 limits the stack depth in order to use "page 0" memory for speed, so this example would have to be modified to run on it.

The <BUILDS ... DOES> construct, which creates the new data type, is one of the most advanced concepts of FORTH. Briefly, the <BUILDS part is executed when SINTABLE is defined; that is, it creates the table. The DOES> part defines what happens when SINTABLE is executed. Once TABLE has been defined, any number of tables of varying length can be declared using the word. Similar definitions can create special-purpose arrays such as word, byte, or bit arrays, user-defined record structures or other data objects, or user-defined classes of operations. [An excellent explanation of the words <BUILDS and DOES> is given in Kim Harris' article "FORTH Extensibility," also in this issue....GW]

```

OK
58 LIST
SCR # 58
0 ( TRIG LOOKUP ROUTINES - WITH SINE *10000 TABLE)
1 : TABLE ( ... N -> . CREATE 'TABLE' DATA TYPE.)
2 <BUILDS 0 DO ; LOOP ( COMPILER N ELEMENTS)
3 DOES) SWAP 2 * + @ ( EXECUTE TABLE LOOKUP)
4 ;

```

Listing 10 continued on page 124



**PERSONAL
COMPUTER
SYSTEMS**

A Warner Communications
Company

ATARI® 800™

List \$1080

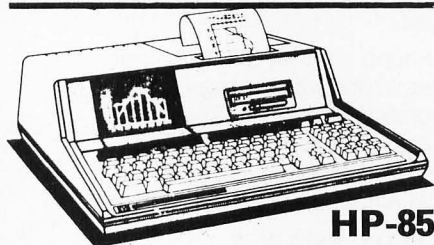
ONLY \$849



ATARI® 400™, List \$630

OUR PRICE ONLY \$499

820 PRINTER, List \$599.95 \$499
810 DISK DRIVE, List \$699.95 \$589



HP-85

- Extended BASIC Language **Call for Price**
- Advance Graphics
- CRT Built-In Display
- Magnetic Tape Cartridge for Storage

CALCULATORS BY

HEWLETT PACKARD

- HP-41C Calculator, "A System" ... \$289.95
- HP-32E Scientific w/Statistics ... \$ 53.95
- HP-33C Scientific Programmable ... 99.95
- HP-34C Advanced Scientific Programmable 123.95
- HP-37E Business Calculator 58.95
- HP-67 Handheld Fully Advanced Programmable Scientific for Business & Engineering 298.95
- HP-97 Desktop w/Built-in Printer .. 579.95

- APPLE II, 16K, List \$1195 \$ 989
- 32K, List \$1395 \$1169
- 48K 1259

COMMODORE PET Call for Prices

Prices do not include shipping by UPS. All prices and offers are subject to change without notice.

**Personal
PC
Systems**



609 Butternut Street
Syracuse, N.Y. 13208
(315) 478-6800

Circle 81 on inquiry card.

Listing 10 continued

```

5 10000 9998 9994 9986 9976 9962 9945 9925 9903 9877
6 9848 9816 9781 9744 9703 9659 9613 9563 9511 9455
7 9397 9336 9272 9205 9135 9063 8988 8910 8829 8746
8 8660 8572 8480 8387 8290 8192 8090 7986 7880 7771
9 7660 7547 7431 7314 7193 7071 6947 6820 6691 6561
10 6428 6293 6157 6018 5878 5736 5592 5446 5299 5150
11 5000 4848 4695 4540 4384 4226 4067 3907 3746 3584
12 3420 3256 3090 2924 2756 2588 2419 2250 2079 1908
13 1736 1564 1392 1219 1045 0872 0698 0523 0349 0175
14 0000 ( 91 ELEMENTS OF TABLE PLACED ON STACK)
15 91 TABLE SINTABLE ( RETURNS SINE, 0-90 DEGREES ONLY)
OK

```

SCR # 59

```

0 ( SINE AND COSINE TABLE-LOOPUP ROUTINES)
1 : S180 ( N -> N , RETURNS SINE, 0-180 DEGREES)
2 DUP 90 > ( IF GREATER THAN 90 DEGREES,)
3 IF 180 SWAP - ENDIF ( SUBTRACT FROM 180)
4 SINTABLE ( THEN TAKE SINE)
5 ;
6 : SIN ( N -> SINE, RETURN SINE OF ANY NUMBER OF DEGREES)
7 360 MOD ( BRING WITHIN + OR - 360)
8 DUP 0< IF 360 + ENDIF ( IF NEGATIVE, ADD 360)
9 DUP 180 > ( TEST IF GREATER THAN 180)
10 IF 180 - S180 MINUS ( IF SO, SUBTRACT 180, NEGATE SINE)
11 ELSE S180 ENDIF ( OTHERWISE, STRAIGHTFORWARD)
12 ;
13 : COS ( N -> COSINE,)
14 360 MOD ( PREVENT OVERFLOW NEAR 32,767)
15 90 + SIN ; ( COSINE IS SINE WITH 90 DEGREES PHASE SHIFT)
OK

```

```

OK
58 LOAD 59 LOAD OK
# SIN , 0 OK
# COS , 10000 OK
90 SIN , 10000 OK
45 SIN , 7071 OK
1 SIN , 175 OK
361 SIN , 175 OK
1000 SIN , -9848 OK
10000 SIN , -9848 OK
10000 COS , 1736 OK
-25281 COS , 1564 OK
32767 SIN , 1219 OK
32767 COS , 9925 OK
-1 SIN , -175 OK
: SINSCALE ( N DEGREES -> N , SCALE BY SINE)
SIN 10000 * / ( MULTIPLY, THEN DIVIDE; 32 BITS INTERMEDIATE)
; OK
100 45 SINSCALE , 70 OK
10000 45 SINSCALE , 7071 OK
30000 -5 SINSCALE , -2616 OK

```

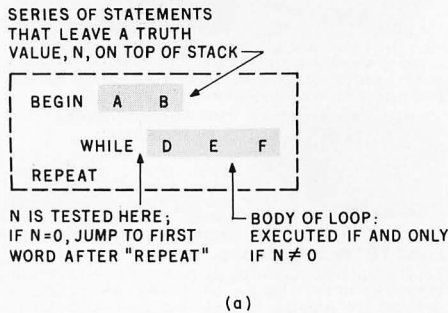
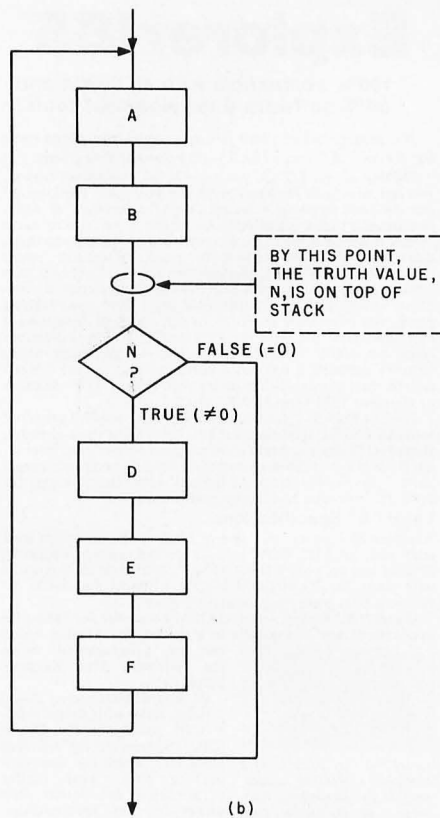



Figure 5: An explanation of the BEGIN ... WHILE ... REPEAT construct. As shown in figure 5a, the FORTH words between BEGIN and WHILE perform operations that leave a truth value, N, on top of the stack. The value of N determines whether the body of the loop (the words between WHILE and REPEAT) is performed or not. The loop repeats until N evaluates to false (N=0). Figure 5b gives the equivalent construct in conventional flowchart notation.



4. Hicks, S M, "FORTH's Forte is Tighter Programming," *Electronics*, March 15, 1979, pages 114 thru 118.
5. James, J S, "FORTH for Micro computers," *Dr Dobb's Journal of Computer Calisthenics & Orthodontia*, May 1978; reprinted in *SIGPLAN Notices* (Special Interest Group on Programming Languages of the Association for Computing Machinery), October 1978.
6. Meinzer, K, "IPS, an Unorthodox High-Level Language," *BYTE*, January 1979, pages 146 thru 159.
7. Moore, C H, "FORTH: a New Way to Program a Computer," *Astronomy and Astrophysics Supplement*, 1974, number 15, pages 497 thru 511.
8. Moore, C H, and E D Rather, "The FORTH Program for Spectral Line Observing," *Proceedings of the IEEE*, September 1973, pages 1346 thru 1349.
9. Rather, E D, and L Brody, *Using FORTH* (2nd rev ed) FORTH Inc, Hermosa Beach CA, 1980.
10. Rather, E D, and C H Moore, "The FORTH Approach to Operating Systems," *ACM 1976 Proceedings*, Association for Computing Machinery, 1976.
11. Sachs, J, *An Introduction to STOIC*, Technical Report BMEC TR001, Harvard-MIT Program in Health Sciences and Technology, Cambridge MA, June 1976.
12. Stein, P, "The FORTH Dimension: Mini Language Has Many Faces," *Computer Decisions*, November 1975, page 10.
13. Stevens, W R, *A FORTH Primer*, Kitt Peak National Observatory, Tucson AZ, 1979.
14. Taylor, A, "FORTH Becoming Hothouse for Developing Languages," *Computerworld*, July 30, 1979.
15. Taylor, A, "FORTH Setting Coding Trend?," *Computerworld*, August 13, 1979.
16. Taylor, A, "Trade Language Families Can Sprout from FORTH," *Computerworld*, August 27, 1979.
17. Wells, D C, "Interactive Image Analysis for Astronomers," *Computer*, August 1977, pages 30 thru 34.

SELECTED BIBLIOGRAPHY

1. Bartoldi, P, "Stepwise Development and Debugging Using a Small Well-Structured Interactive Language for Data Acquisition and Instrument Control," *Proceedings of the International Symposium and Course on Mini- and Microcomputers and their Applications*, Acta Press, Anaheim CA, 1976, pp 117-122.
2. Ewing, M S, *The Caltech FORTH Manual*, California Institute of Technology, Pasadena CA, 1978.
3. Forsley, L, *URTH Tutorial*, University of Rochester, Rochester NY.

S-100 USERS: GIVE YOUR COMPUTER THE GIFT OF SIGHT!

The DS-80 Digisector[®] is a random access video digitizer. It works in conjunction with a TV camera (either interlaced or non-interlaced video) and any S-100 computer conforming to the IEEE standards. Use it for:

- Precision Security Systems
- Moving Target Indicators
- Computer Portraiture
- Fast To Slow Scan Conversion
- Robotics
- Reading UPC Codes, schematics, paper tape, musical scores



● IMAGE PROCESSED BY DS-80 ●

CHECK THESE FEATURES:

- High resolution** — a 256 x 256 picture element scan
- Precision** — 64 levels of grey scale
- Speed** — Conversion time of 14 microseconds per pixel
- Versatility** — scanning sequences user programmable
- Economy** — a professional tool priced for the hobbyist; comes fully assembled, tested and burned in, with fully commented portrait printing software.

Price: \$349.95 MasterCharge and Visa

THE MICRO
WORKS

P.O. BOX 1110, DEL MAR, CA 92014 714-756-2687

Text continued from page 10:

You should also look at FORTH if you have limited computer or financial resources. FORTH is a *big* language in a *small* package, and you can buy a version of FORTH for as little as \$20. (See "Selected FORTH Vendors," on page 98.) Unlike most new languages that gobble up more and more of the 64 K bytes allotted to an 8-bit microcomputer (some won't comfortably fit in 64 K bytes), there is plenty of room for very large FORTH programs even in a 16 K machine. FORTH takes up only about 8 K bytes, and this can be pared down; in an industrial application that will run only one program, the FORTH interpreter can be made as small as 800 bytes. Also, FORTH can be run on cassette-based systems due to its small size; although this is still more inconvenient than running FORTH on a disk system, most languages that use a disk are impractical or impossible on cassette-only systems.

Finally, you may want to consider FORTH for applications where speed is of the utmost importance. Since portions (or all) of a FORTH program can be written in the assembly lan-

guage of the host computer, FORTH programs can be written that compare favorably in speed with machine-language programs. And, again, productivity is higher using FORTH than it is with machine language.

What Is a Threaded Language?

Imagine a language that starts with a few fundamental subroutines written in the machine language of the host computer; eg: routines to put a character to the display device, to get a character from the keyboard, to multiply two fixed-point numbers. Then imagine that the only way to combine these subroutines is to string them together (with embedded data bytes) as a series of subroutine calls; eg: a routine to get a signed multidigit number from the keyboard is written as a controlled series of calls to the subroutine that gets a character. Then these routines are called by other routines that perform even bigger tasks. For example, a routine to sum a series of signed numbers entered from the keyboard is written as series of subroutine calls that includes the one mentioned just above. The final pro-

Special Notation Used in This Issue

Because FORTH is such an unusual language (it uses punctuation marks by themselves and within words), a pair of braces, { }, is sometimes used to set apart FORTH words from the rest of the text. Braces are used under the following conditions:

- When the material being quoted is a series of FORTH words; eg: { 26 LOAD } ;
- When the FORTH word is or contains any of the following punctuation marks: period, comma, colon, question mark, exclamation point, single quote mark, or double quote mark. Two examples are { . } and { (") }.

In addition, spaces are always used to separate FORTH words from other words or punctuation—even when this means doing something like "...the words BEGIN , WHILE , and REPEAT are all..." (spaces between FORTH words and the commas that follow them). There are two reasons for doing this: first, for clarity; and second, to emphasize that the FORTH word in question does not include the punctuation that follows. Some FORTH words do contain punctuation (eg: { IF , }), but such words will always be enclosed in braces (except within program listings).

gram in such a *threaded language* is a series of calls to lower and lower subroutines, dipping repeatedly into machine-language routines under the control of higher-level routines. The addresses in each subroutine that point to the subroutine or machine language under it make up a "thread" of control that runs through the entire program.

FORTH has so far been implemented as a threaded language. *Threadedness is a language implementation technique, not an inherent quality of any language;* SNOBOL and FORTRAN compilers have been written using threaded code.

Computer Hardware Professionals

Our clients, highly successful manufacturers and OEMs of Computer Systems, Electronic Systems, and Peripherals, have immediate openings for Hardware Development Professionals to work on FUTURE SYSTEMS PROJECTS. Such projects include COMPUTER ARCHITECTURE, DATA COMMUNICATIONS, PERIPHERAL DEVELOPMENT, and POWER SUPPLY DESIGN. Specific openings currently exist at Senior and Intermediate levels for:

COMPUTER ARCHITECTS — Definition and development of Micro- Mini-computer systems.

POWER SUPPLY DESIGN ENGINEERS — Switching regulators for Off-Line Power supplies. Experience in High Frequency P.W.M. techniques and AC Power Distribution would be desirable.

MICROPROCESSOR DESIGN ENGINEERS — Design/Development of state-of-the-art Microprocessor based systems and interfaces. Experience on any Microprocessor acceptable.

LSI DESIGN DEVELOPMENT — Numerous positions with local systems oriented firms in LSI technology development.

CPU DESIGN ENGINEERS — BSEE/BSCS and/or experience in the design of Digital Computers or Microprocessor systems. Requires an understanding of Software, i.e. ASSEMBLY, FORTRAN, or PL-1.

DIGITAL LOGIC AND CIRCUIT DESIGN ENGINEERS — Logic and Circuit design plus a familiarity with TTL, CMOS, LSI/VLSI, etc.

ANALOG DESIGNERS — 30 to 40 megahertz Phase Lock loop experience. Experience with 80 megahertz power drivers and DC motors.

PCB DESIGNERS — With CAD experience.

COMMUNICATIONS SYSTEMS DEVELOPERS — Experience with store and forward message switching, Network Data Link Control, and/or PBX and EPX Systems.

Compensation on all positions ranges from low 20's to low 40's, based upon experience. Client companies are equal opportunity/affirmative action employers, provide excellent benefits, and assume all fees.

Qualified applicants will receive IMMEDIATE RESPONSE and are invited to contact: Don Bateman, in strict confidence, at (617) 861-1020. Or submit current resume to him for review. For those who find it inconvenient to call during working hours, our office will be open until 7:30 p.m.

Contact: Don Bateman

rk Robert Kleven and Co., Inc.
INDUSTRIAL RELATIONS MANAGEMENT CONSULTANTS

Three Fletcher Avenue, Lexington, Massachusetts 02173
Telephone (617) 861-1020



Member
Massachusetts Professional Placement Consultants
National Computer Associates
Officers, Nationwide
Representing Equal Opportunity Employers M.F.

FORTH: Pro and Con

Pros: I have already mentioned most of the advantages of FORTH. The language is:

- Compact;
- Fast, although this is due to its implementation in threaded code, not its inherent qualities;
- Structured: it has the major constructs of structured programming and, in fact, does not have any kind of *goto* statement, thus forcing it to be structured;
- Extensible;
- Highly portable.

These last two features deserve further description. The *extensibility* of FORTH is probably its most important feature. Never before in a high-level language has it been so easy to add new features, new data types, and new operators to a language. Unlike other languages, these new words (everything in FORTH is called a *word*) have the same priority and receive the same treatment as words defined in the standard FORTH vocabulary. For example, you can

define a word 10+ that will add ten to any number it is given; or, in fact, you can even redefine the addition operator +. You can also define entirely new families of words in FORTH. This advanced topic is ably discussed in what I believe is the only written treatment of the subject *anywhere* in FORTH literature by Kim Harris in his article, "FORTH Extensibility," on page 164.

Most FORTH programs can be transferred from, say, a mainframe computer to a microcomputer without modification; therefore, FORTH is *highly portable*. Most of the FORTH words supplied in a given system have been defined to do the same operation regardless of the computer used. Although the vocabulary of words varies from supplier to supplier, most FORTH programs will run with minor or no modifications. A standard set of words, called FORTH-79, collectively developed by many of the major suppliers and users of FORTH, will help in this situation.

Cons: Here are some of the disadvantages of FORTH:

- FORTH code is hard to read.

This is probably the most common complaint against the language. As a new user, I can say that you slowly get used to the odd syntax of the language. The stack architecture (see below) of the language contributes to the novice's initial disorientation, but this feeling is usually blamed on the unreadability of the language. In addition, the stack architecture encourages the storage of working values on the stack rather than in variables with names. Variable names, if chosen properly, give vital clues to the workings of a program; this scarcity of variable names makes most FORTH programs less readable. Adequate indentation and comments can help a FORTH program, but programmers of FORTH, like programmers of all other languages, often omit these aids to comprehension.

● The stack architecture of FORTH offers disadvantages as well as advantages. Remember the odd feeling you got the first time you used a Hewlett-Packard calculator and had to punch in "5 ENTER 3 + " instead of the more understandable "5 + 3 = " ? FORTH uses the same *reverse Polish notation* (abbreviated RPN), where the objects being entered come *before* the operators that work on them.

Not only does this take some getting used to (it takes even longer before you can fluently "think in FORTH"), it also encourages a scarcity of named variables, as mentioned above. In addition, stack-manipulating words like SWAP, DUP (for duplicating the top entry on the stack), ROT (for rotating the top three items on the stack), and others muddle the FORTH program and make it hard to tell just what variable is being operated on. This uncertainty is particularly evident during debugging; most of your time is spent finding out why what you *thought* was on the stack isn't there.

● FORTH encourages programming "tricks" in place of plain, easier to read programming. Although the examples to support this statement have already been mentioned, I think the statement as a generality is true. We must remember that, especially since lack of memory is usually *not* a problem in FORTH, FORTH programmers should name appropriate variables and, in general, worry less about fitting a program on one screen (a basic unit of FORTH program-

TRS-80, PET, APPLE, SORCERER Communications Interface Systems



- Send & Receive Morse Code / Radioteletype
- Teaches Morse Code! / Copies wire services!
- Complete Hardware & Software Package
- Extensive User Manuals
- From \$129

Write or call for complete catalog

MACROTRONICS, inc.®

1125 N. Golden State Blvd. / Suite G
Turlock, CA 95380 (A)
(209) 667-2888 / 634-8888

California residents add 6% tax

We are experiencing telephone difficulties, please keep trying.



ming) and more about making it readable.

However, drawing a comparison to APL, any language that compresses a lot of program into a small number of lines suffers from readability problems. Broad, powerful algorithms often represent complex processes; when they are described in a terse notation, they look like programming tricks. In this case, the only remedy is to use a lot of comments. The lack of such comments is solely the fault of the programmer, not of the computer language.

● FORTH lacks many of the programming constructs we are used to—strings, arrays, floating-point numbers—but that's not the whole story. Many applications, for example, can get by without floating-point numbers: look at the number of programs written in Integer BASIC for the Apple II. With a maximum absolute numeric value of 32,767, normal FORTH can handle many problems by simply assuming a decimal point. In addition, all versions of FORTH can *add* all these features and more, simply by defining new words. For example, MMSFORTH, a version

of FORTH for the TRS-80 by Miller Microcomputer Services, has over ten screens (each *screen* is 16 lines of source code) that implement their version of words for double-precision math, arrays, strings, random numbers, and TRS-80 graphics. You compile a series of screens, thus adding to the size of your resident FORTH interpreter, *only if you need these features*. So you *can* have all these programming constructs and tools, but only if you write them yourself or get somebody else to write them for you.

Friends of FORTH

Almost everyone who is working in FORTH professionally is doing good work, but a few people or groups of people deserve special mention. Foremost in this group is Charles H Moore and, through him, the company FORTH Inc. Moore developed the language over a long period of time (see his article "The Evolution of FORTH, an Unusual Language," on page 76) and promoted it through his company FORTH Inc. Elizabeth Rather, who contributed significantly to the

development of the language and who is vice-president at FORTH Inc, should also be mentioned in this context.

Then there is the FORTH Interest Group (POB 1105, San Carlos CA 94070), without whose efforts low-cost versions of FORTH would not be available. Although many people in the group have contributed to its working, names that must be mentioned are Bill Ragsdale (coordinator), Dave Boulton, Kim Harris, John James, and George Maverick. Over the past two years, this group has collectively raised its membership from a few dozen people in northern California to over a thousand members worldwide. In the process, they have also publicized FORTH at numerous conventions and have distributed public-domain versions of FORTH (called fig-FORTH) for all the major microprocessors; ie: 8080, 6800, 6502, 9900, PACE, and LSI-11. Although they supply only listings and documentation, versions customized for various popular microcomputers are available inexpensively. In addition, they are working on standardizing certain extensions to FORTH (floating-point numbers, arrays, etc), and they publish a very professional-looking bimonthly magazine called *FORTH Dimensions*. The group has monthly meetings at the Liberty House Department Store in Hayward, California, on (what else?) the fourth Saturday of each month. Membership in the FORTH Interest Group (which includes a subscription to its magazine) is \$12 per year, \$15 overseas.

A final group that must be mentioned is Miller Microcomputer Services of Natick, Massachusetts, which sells and supports a version of FORTH, called MMSFORTH, and other related FORTH products for the Radio Shack TRS-80 Model I. Not only do they provide a fine version of FORTH with arrays, strings, graphics, and other extensions, they are the only microcomputer-FORTH vendor that supports its product with both information and new vocabularies of FORTH words. (For example, they have a set of FORTH words that add 6- and 15-digit floating-point arithmetic, complex numbers, and a full Z80 assembler, all for \$29.95.) They also publish an *MMSFORTH Newsletter* that always has some

ARTIFICIAL INTELLIGENCE

C/PM® or S-100 SYSTEM

For Your

"SHIVA®" is a highly-sophisticated VIRTUAL-PERSONALITY® multi-level multi-user multi-tasking executive (operating system) for S-100 based systems. It provides your microcomputer system immediately with power comparable to that of large-frame maxi-computers for a remarkably small price, yet SHIVA® requires surprisingly little R.A.M. area, and is conversational!!! SHIVA's® English-like input/output is interactive, dynamic, and may be reconfigured or expanded by the user. And SHIVA® gives you the freedom to expand indefinitely . . . with tremendous hardware and software choice: SHIVA® supports hard disks and floppies . . . R.A.M. addressing beyond 64 kilobytes . . . time-sharing . . . multi-level user-reconfigurable password protection . . . and features shell-commands similar to UNIX® in structure!! SHIVA® is compatible with C/PM® and C/DOS® for easy implementation and near universal software support!!! SHIVA® is available for 8080, 8085, MC6800, 6502, and Z80®-based systems.

Versions are in development for ZILOG Z8000® 16-BIT, INTEL 8086® and INTEL 88002® 32-BIT PROCESSORS . . .


And Omega Research® is dedicated to non-obsolescence and system superiority in software choice . . . SHIVA® supports BASIC, FORTRAN, COBOL, a MACRO-ASSEMBLER, DATA BASE MANAGEMENT, ALGOL-60, PASCAL . . . interfaces in development for UNIX®, C, LISP, PL/I, APL, and RT-11®.

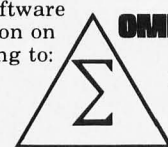
And needless to say, SHIVA® is very fast

SHIVA® \$350 --- Available on 8" I.B.M. Soft-Sector'd Disks and 5" C/DOS® (Cromemco) Diskettes. Includes complete Documentation. . . .
M.C. & Visa orders accepted

"SHIVA", "VIRTUAL-PERSONALITY", and "OMEGA RESEARCH" are trademarks of OMEGA RESEARCH.
"RT-11" is a trademark of DIGITAL EQUIPMENT CORPORATION.
"UNIX" is a trademark of BELL LABORATORIES
"CP/M" is a trademark of DIGITAL RESEARCH OF CALIFORNIA
"C/DOS" is a trademark of CROMEMCO, Inc.
"Z-80" and "Z-8000" are trademarks of ZILOG, Inc.
"INTEL" is a trademark of INTEL CORPORATION

No shipments prior to return of signed software license agreement. For detailed information on "SHIVA®," send \$1.00 postage and handling to:


CALIFORNIA RESIDENTS ADD 6% SALES TAX


OMEGA RESEARCH
P.O. Box 479
Linden, Ca. 95236
(209) 334-6666
9am to 5pm Mon.-Fri.

goodies you'd expect to pay money for. The people at MMSFORTH are A Richard (Dick) Miller and Judy Miller, along with free-lance programmer Tom Dowling, who wrote MMSFORTH for the TRS-80.

In addition, the major vendors of FORTH should be commended for the way they have worked and are working together to help standardize the language. The people mentioned above, along with the European FORTH Users' Group (EFUG), have met as the International FORTH Standards Team to work out a standard set of FORTH words (with standard behavior) that can be used to increase the already high portability of FORTH programs. Once the proposed FORTH-79 standard is approved by this standards team, FORTH Inc, the FORTH Interest Group, and Miller Microcomputer Services have indicated that they will bring out new FORTH versions conforming to this standard.

Variants of FORTH

A few other FORTH-like languages should be mentioned here. URTH (University of Rochester Threaded

language) is simply FORTH by another name. I am told that CONVERS, an experimental language that was offered by the Digital Group, is a FORTH-like language.

STOIC is a language that is different from FORTH primarily in some small syntax rules, although its enthusiasts claim it is more powerful than FORTH. From reading the documentation, I have found that STOIC interacts differently and has more sophisticated disk access than FORTH. CP/M Users Group (1651 Third Ave, New York NY 10028) distributes STOIC on two 8-inch single-density CP/M floppy disks; the cost is \$20, which includes postage, documentation (on CP/M DOC files), and group membership fees. STOIC was developed by Roger G Mark and Stephen K Burns in the Biomedical Engineering Center for Clinical Instrumentation, funded by the Harvard-MIT Program in Health Sciences and Technology in Cambridge, Massachusetts.

Also, I am very excited about a book nearing publication: *Threaded Interpretive Languages* by Ron Loeliger. This book, to be published

soon by BYTE Books, delves deeper into the practical aspects of designing and implementing a threaded language than any book I have seen. Not only does it demonstrate exactly how the machine code must work, it also details the specific implementation of ZIP (which looks like FORTH under another name) in Z80 assembly language. The book promises to be *the* definitive work on how threaded languages perform.

Final Notes

As we received more and more FORTH articles, I realized that we would soon have too many for this special August issue. I immediately scheduled for subsequent nontheme issues those extra articles we could not use at this time, a process known as "holding down the FORTH." In any case, we have several FORTH articles that will appear in upcoming issues of BYTE. These include an article on recursion in FORTH by George Flammer, a tutorial on string-manipulating FORTH words by John Cassady, a history of the FORTH Standards Team by Bill Ragsdale, and a detailed discussion of the different kinds of threaded codes by Terry Ritter and Gregory Walker.

We hope you will enjoy looking at the FORTH tapestry presented in this issue. ■

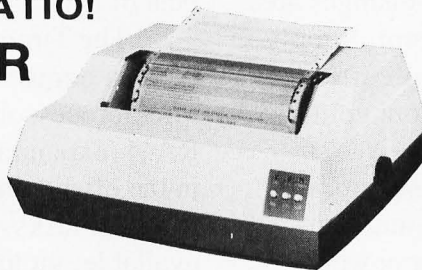
**NOBODY CAN MATCH OUR
DOLLAR/QUALITY RATIO!**

MS-204 PRINTER

INTRODUCTORY PRICE:

\$795

CABLE: \$34.50



**Compatible with TRS-80, Apple, Pet
or any other Centronics-type system**

Features

- 132/80 Columns, 63 LPM, Bi-Directional, Nominal Thruput
- 100% Heavy Duty Cycle - High Reliability, 100 Million Character Print Head Life
- Sprocket Feed; Variable Forms Width, 2.5" - 9.5"
- Double Width Characters: 40,66 Characters per line
- 9 x 7 Dot Matrix Character Font
- 6-Channel Electronic Vertical Format Unit
- Documentation Included

Ask about our 8-inch Drives & Software

**MATCHLESS
SYSTEMS**

18444 S. Broadway
Gardena, CA 90248
(213) 327-1010

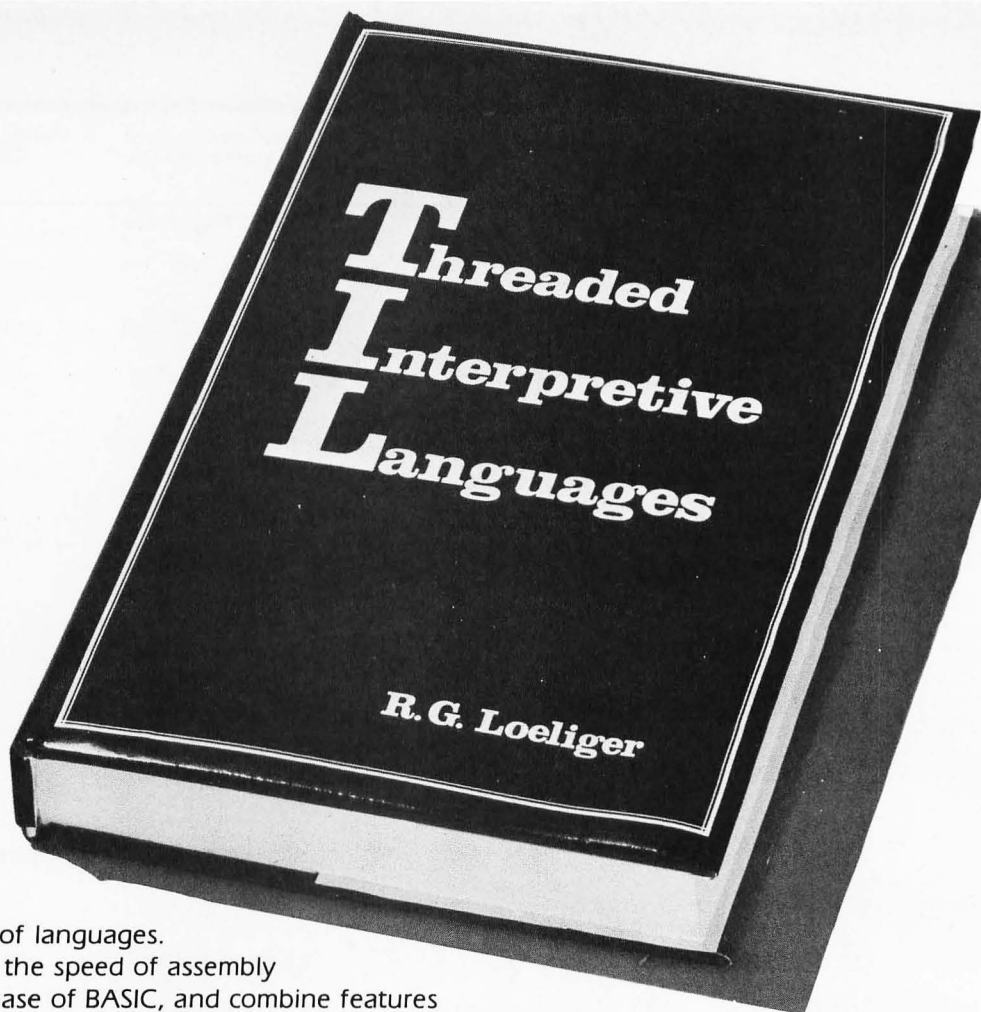
© 1980 Matchless Systems & MarketPlan

Articles Policy

BYTE is continually seeking quality manuscripts written by individuals who are applying personal computer systems, designing such systems, or who have knowledge which will prove useful to our readers. For a more formal description of procedures and requirements, potential authors should send a large (9 by 12 inch, 30.5 by 22.8 cm), self-addressed envelope, with 28 cents US postage affixed, to BYTE Author's Guide, 70 Main St, Peterborough NH 03458.

Articles which are accepted are purchased with a rate of up to \$50 per magazine page, based on technical quality and suitability for BYTE's readership. Each month, the authors of the two leading articles in the reader poll (BYTE's Ongoing Monitor Box or "BOMB") are presented with bonus checks of \$100 and \$50. Unsolicited materials should be accompanied by full name and address, as well as return postage.

Anatomy of a Threaded Language



Threaded languages (such as FORTH) are an exciting new class of languages. They are compact and fast, giving the speed of assembly language with the programming ease of BASIC, and combine features found in no other programming languages. An increasing number of people are using them, but few know much about how they work. Is a threaded language interpreted or compiled? How much memory overhead does it require? Just what is an "inner interpreter"? **Threaded Interpretive Languages**, by R. G. Loeliger, concentrates on the development of an interactive, extensible language with specific routines for the ZILOG Z80 microprocessor. With the core interpreter, assembler, and data type defining words covered in the text, it is possible to design and implement programs for almost any application imaginable. Since the language itself is highly segmented into very short routines, it is easy to design equivalent routines for different processors and produce an equivalent threaded interpretive language for other development systems. If you are interested in learning how to write better FORTH programs or you want to design your own powerful, but low-cost, threaded language specific to your needs, this book is for you.



This and other BYTE/McGraw-Hill books are available from BYTE Books or your local computer store.

ISBN 0-07-038360-X
Price \$18.95

Please send _____ copies of **Threaded Interpretive Languages**

Name Title Company

Street City State/Province Code

Check enclosed in the amount of \$ _____
 Bill Visa Bill Master Charge
Card No. _____

Exp. Date. _____

Add 75¢ per book to cover postage and handling.
Please remit in U.S. funds or draw on a U.S. Bank.

Available in Nov. 1980



70 Main St.
Peterborough, NH 03458

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

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

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

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. 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.)

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-

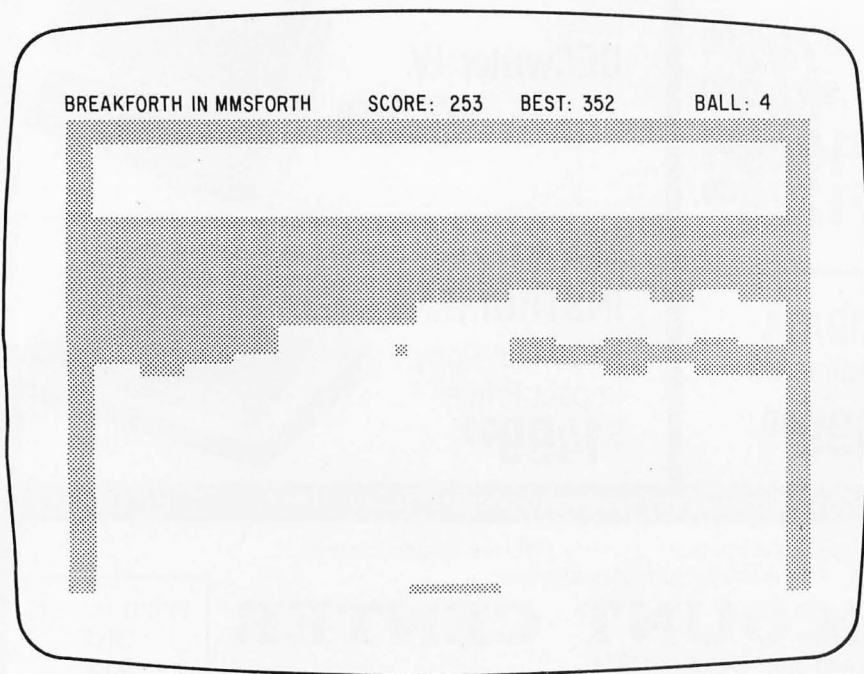


Figure 1: One view of the TRS-80 video screen during a BREAKFORTH game.

MORE FOR YOUR RADIO SHACK TRS-80 MODEL I !

- ★ **MORE SPEED**
10-20 times faster than Level II BASIC.
- ★ **MORE ROOM**
Compiled code plus VIRTUAL MEMORY makes your RAM act larger.
- ★ **MORE INSTRUCTIONS**
Add YOUR commands to its large instruction set!
Far more complete than most Forths: single & double precision, arrays, string-handling, more.
- ★ **MORE EASE**
Excellent full-screen Editor, structured & modular programming
Optimized for your TRS-80 with keyboard repeats, upper/lower case display driver, single- & double-width graphics, etc.
- ★ **MORE POWER**
Forth operating system
Interpreter AND compiler
Internal 8080 Assembler (Z80 Assembler also available)
VIRTUAL I/O for video and printer, disk and tape (10-Megabyte hard disk available)

mmsFORTH

THE PROFESSIONAL FORTH FOR TRS-80

Prices:
MMSFORTH Disk System V1.9 (requires 1 disk drive & 16K RAM) just **\$79.95***
MMSFORTH Cassette System V1.8 (requires Level II BASIC & 16K RAM) **\$59.95***

AND MMS GIVES IT PROFESSIONAL SUPPORT

Source code provided
MMSFORTH Newsletter
Programming staff available
Many demo programs aboard
MMSFORTH User Groups

FLOATING POINT MATH (L2 BASIC ROM routines plus Complex numbers, Rectangular-Polar coordinate conversions, Degrees mode, more), plus a full Z80 ASSEMBLER; all on one diskette... **\$29.95***
THE DATAHANDLER, a very sophisticated database management system operable by non-programmers (requires 1 drive and 32K RAM); with manuals **\$59.95***

Other packages under development

FORTH BOOKS AVAILABLE

MICROFORTH PRIMER — comes with MMSFORTH; separately **\$15.00***
USING FORTH — more detailed and advanced than above **\$25.00***
URTH TUTORIAL MANUAL — very readable intro. to U/Rochester Forth **\$19.95***
CALTECH FORTH MANUAL — good on Forth internal structure, etc **\$6.95***

* — Software prices are for single-system user license and include manuals. Add \$2.00 S/H plus \$1.00 per additional book; Mass. orders add 5% tax. Foreign orders add 15%. UPS COD, VISA & M/C accepted; no unpaid purchase orders, please.

Send SASE for free MMSFORTH information.
Good dealers sought.

MMSFORTH is available from your computer dealer or
MILLER MICROCOMPUTER SERVICES (B1)

61 Lake Shore Road, Natick, MA 01760
(617) 653-6136

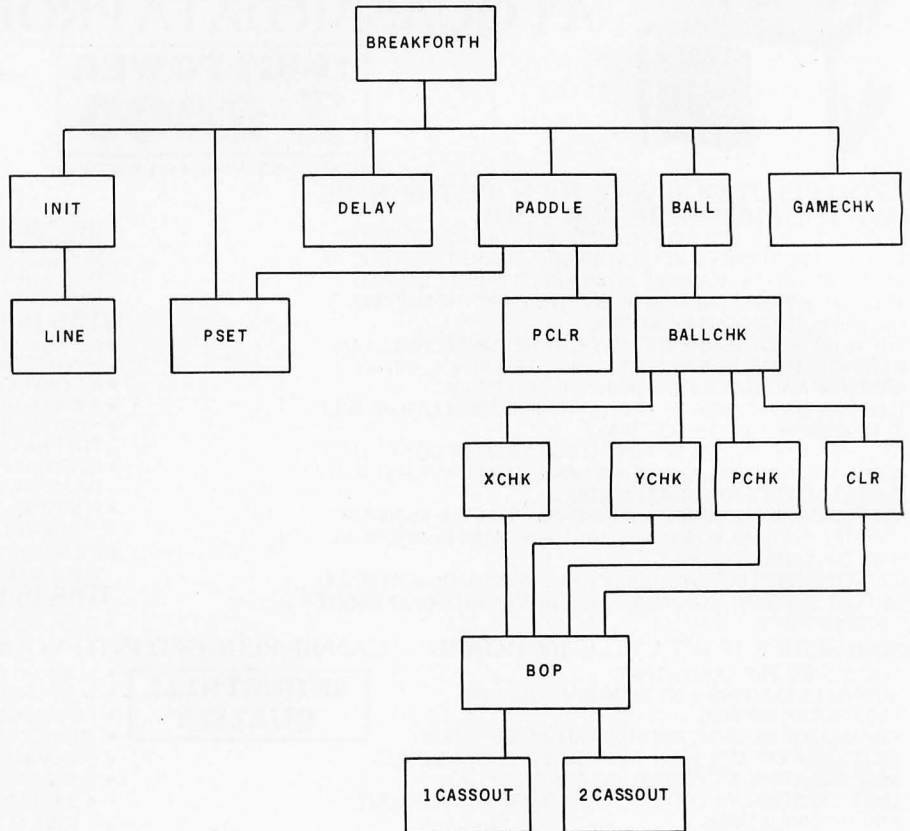


Figure 2: A hierarchical diagram of the BREAKFORTH program. Each box contains a word used within the BREAKFORTH program and is used by the word(s) in the box(es) above it. See table 1 for a definition of each word.

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

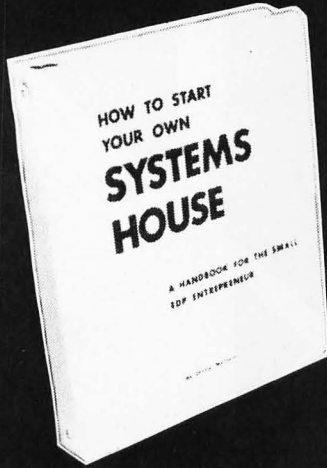
clear-to-the-end-of-the-line signal on the TRS-80. (By the way, the 30 is the decimal number thirty; although you can change to hexadecimal with the word HEX or to any other numeric base, MMSFORTH assumes decimal numbers unless told otherwise.)

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

YOU TOO can become a successful computer ENTREPRENEUR!



HOW TO START YOUR OWN SYSTEMS HOUSE is a practical step-by-step guide for the EDP professional or small businessman who wants to enter the micro-computer systems business.

Written by the founder of a successful systems house, this fact-filled 220-page manual covers virtually all aspects of starting and operating a small systems company. It is abundant with useful, real-life samples: contracts, proposals, agreements and a complete business plan are included in full, and may be used immediately by the reader.

Proven, field-tested solutions to the many problems facing the small systems house are presented.

From the contents:

- New Generation of Systems Houses • The SBC Marketplace • Marketing Strategies • Vertical Markets & IAPs • Competitive Position/Plans of Major Vendors • Market Segment Selection & Evaluation • Selection of Equipment & Manufacturer • Make or Buy Decision • Becoming a Distributor • Getting Your Advertising Dollar's Worth • Your Salesmen: Where to Find Them • Product Pricing • The Selling Cycle • Handling the 12 Most Frequent Objections Raised by Prospects • Financing for the Customer • Leasing • Questions You Will Have to Answer Before the Prospect Buys • Producing the System • Installation, Acceptance, Collection • Documentation • Solutions to the Service Problem • Protecting Your Product • Should You Start Now? • How to Write a Good Business Plan • Raising Capital

6th edition, March 1980 220 pages

Essex Publishing Co. DEPT. 3
285 Bloomfield Avenue Caldwell, N.J. 07006

I would like to order **HOW TO START YOUR OWN SYSTEMS HOUSE** at \$36.00 (New Jersey residents add 5% sales tax)

Check Enclosed VISA Mastercharge

Name _____

Address _____

City _____

State _____ Zip _____

Card # _____ exp. _____

For immediate shipment on credit card orders call (201) 783-6940

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 CARIABLE SPEED 0 CARIABLE 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

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

```

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

```

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.

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

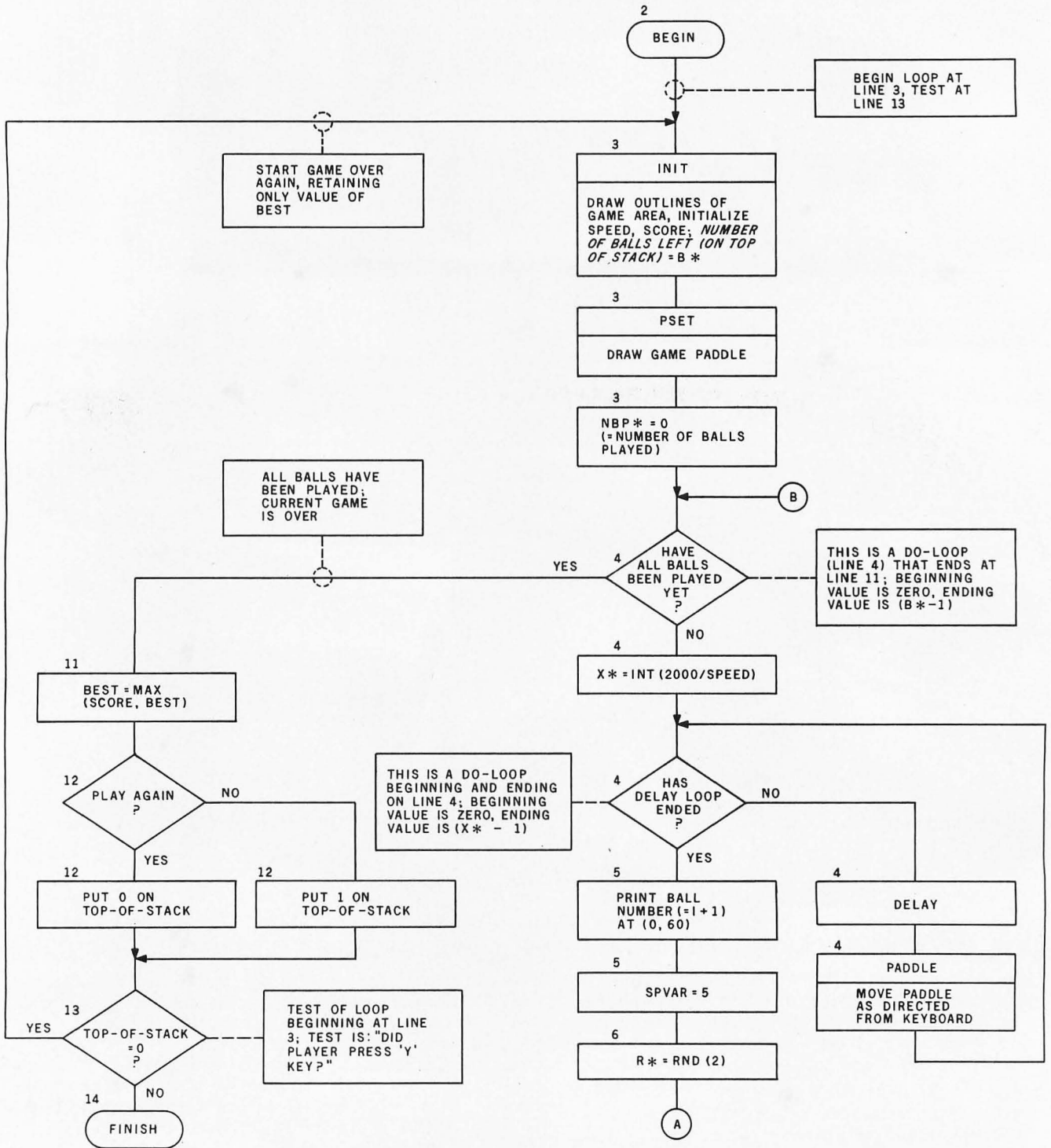


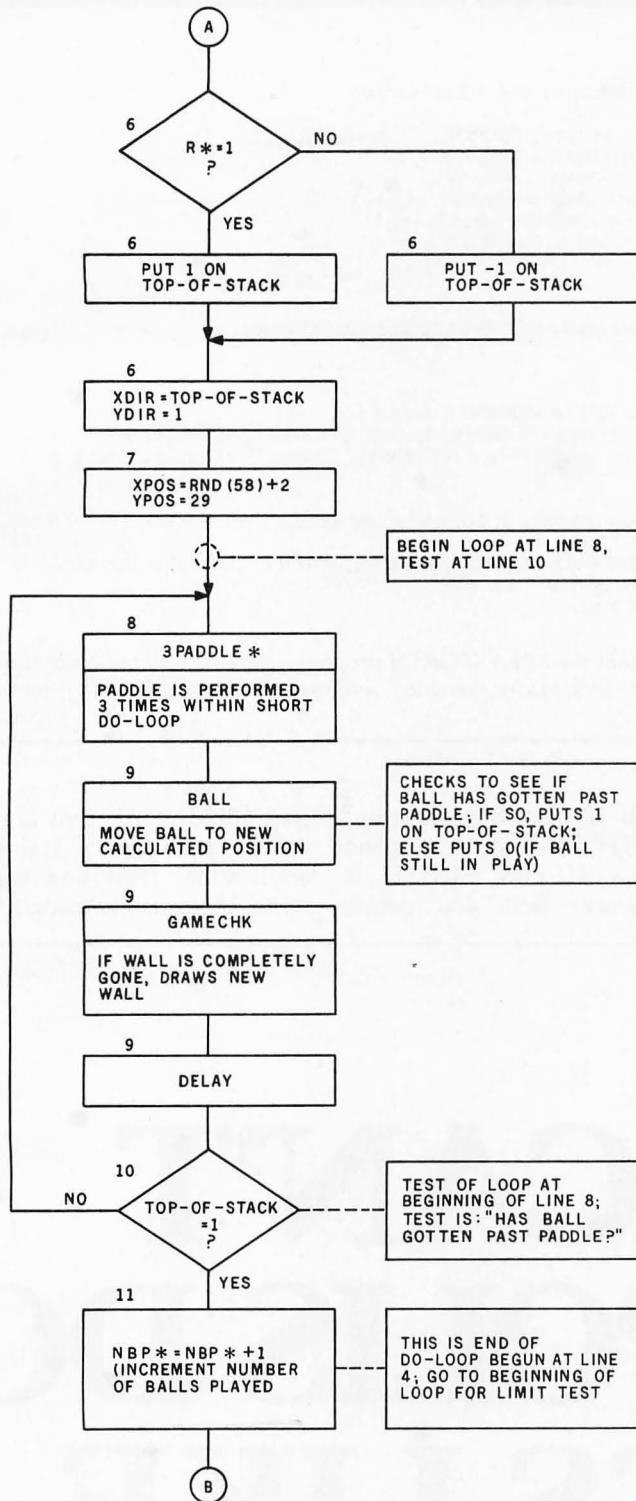
Figure 3: A flowchart for the BREAKFORTH program (given in listing 1, block 55). The number above each box is the line number within block 55 that performs the action of the box. Many calculations in FORTH are done on the stack and do not acquire variable names. Because of this, an asterisk in a variable 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 <i>n</i> on top of stack; moves cursor to line <i>n</i> , 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 , word (which is also the program) specialized words that are helpful in solving problems of a given class or defines one last word (DELAY , in BREAKFORTH . This is a good demonstration of how FORTH is application, then you use them to line 1), then puts all the words defined so far together to define the meant to work: first you define 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*

To be honest, we could. But our customers have come to expect a lot more from us.

They've come to appreciate our desire to innovate, to improve upon, to blaze new trails in floppy disk technology. That's how we got our reputation as the industry's undisputed technological leader.

96 TPI is nothing new for us.

Consider the current hubbub about "new" 96 TPI disk drives. You should know that what may be new to our competition is anything but new to us.

After all, we brought the 100 TPI MegaFloppy™ disk drive to the marketplace more than two years ago. And we've delivered more than 50,000 drives already.

To us, a 96 TPI drive is no big deal. So for the customer who's looking for a double track drive offering compatibility with 48 TPI drives, Micropolis can deliver.

Think of us as double headquarters.

We should also mention that our double track disk drives give you all the storage capacity of an 8-inch floppy in the body of a 5¼-inch floppy. And with our double head version, you get up to 1.2 megabytes. That's more than ten times the capacity of other 5¼-inch floppies.

But our innovations don't stop there. Over the years, many of our ideas have gone on to become

industry standard. And many more will.

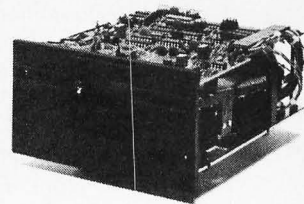
Things like stainless steel, precision-ground lead screws instead of cheaper, less reliable plastic positioners.

We also developed a special disk centering mechanism that is the most accurate in the industry.

And who do you think successfully adapted Group Code Recording technology to the floppy disk drive industry? None other than Micropolis.

Remarkable as our technical achievements may be, some people still wonder how we got to be number two so rapidly in such a fiercely competitive business.

Obviously, we did it by design.



MICROPOLIS™

Where the 5¼-inch OEM drive grew up.

Micropolis Corporation, 21329 Nordhoff Street, Chatsworth, CA 91311. For the telephone number of your nearest OEM rep, call (213) 709-3300.

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.

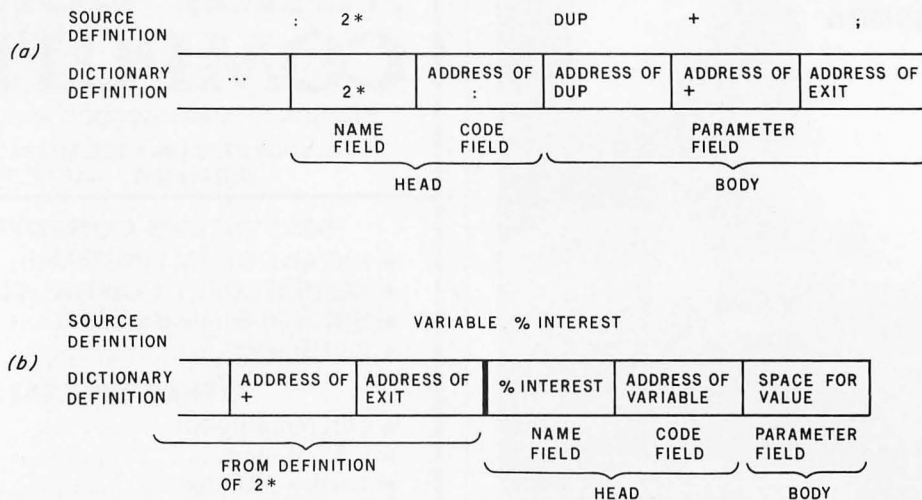


Figure 2: Examples of extending the FORTH language. The first source line adds a new operator named 2* (see figure 2a); the second source line adds a new operand named %INTEREST (see figure 2b).

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 re-compiling 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.

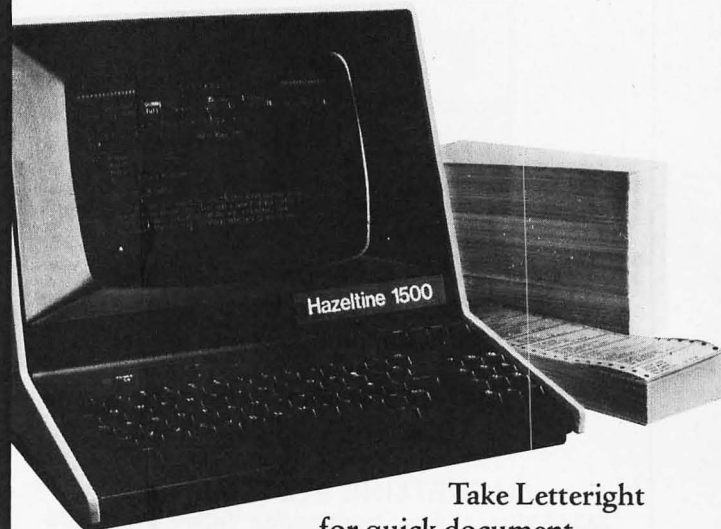
Levels of Extensibility in FORTH

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.

SSG Writing and Mailing Systems.



Take Letterright for quick document preparation and edit plus NAD Name And Address for extensive mailing list capabilities.

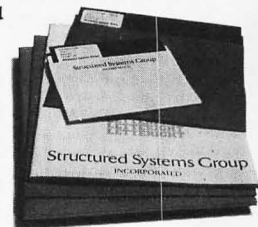
Put them together and you've got a flexible, powerful solution to big and small correspondence problems.

With Letterright you create and edit your document right on the screen. It's much easier to use than a typewriter. The letters are always perfect, and revisions are a snap.

Letterright's "wild card" slots let you create standard letters and forms, then insert information selected from your mailing list to address and "personalize" the letter.

The NAD system will store lots of names and addresses, with identifying information you create. You then print lists, labels, or envelopes of virtually any group you want from the list, or the whole list.

This pair should be working for every microcomputer owner.



Letterright and NAD are part of a full line of working software solutions from Structured Systems Group, all ready to run on any CP/M® microcomputer system. CP/M is a registered trademark of Digital Research.

Structured Systems

5204 Claremont Oakland, Ca 94618 (415) 547-1567
Circle 110 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

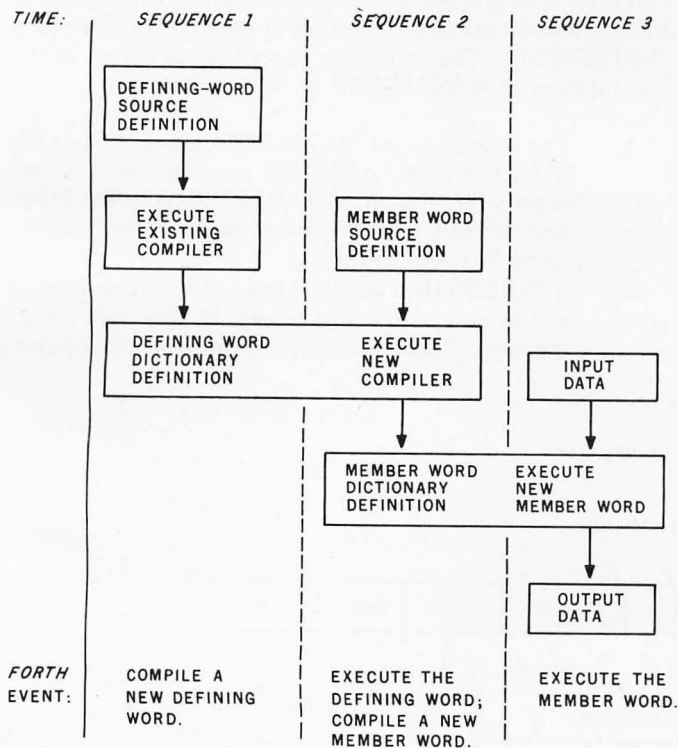


Figure 3: The order of events governing defining words. The first event creates a word that will define a new family of words; this family currently has no members. The second event uses this new family-defining word to create a new family member, a named FORTH word. The third event occurs when any named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

The Working Analyst.



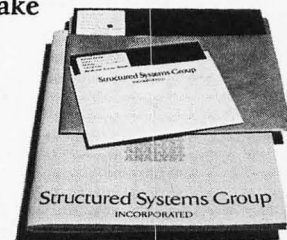
If you would like to put a computer to work collecting, organizing, and summarizing the information you need to make better decisions, take a look at Analyst.

Analyst is a software package designed to let you store and analyze virtually any information involving numbers, dollars, dates, and descriptions. Simply tell Analyst what kind of information

you want to store. Analyst creates a computerized file for that information. And Analyst creates an information entry program for your file that asks you for each entry, and checks your data for errors. (You can create any number of different files.)

Then tell Analyst what reports you want from your data file. There are all sorts of record selection and report formatting options, so you can design an unlimited variety of reports to focus on different aspects of the same data file.

Analyst is so flexible, you'll find a million ways to use it. It is easy to use, so you don't need to be a programmer to make your computer really work for you. If this bit of information intrigues you, find out the rest. You'll like what you see.



Analyst is a part of a full line of working software solutions from Structured Systems Group, all ready to run on any CP/M* microcomputer system. For more information, see your computer retailer, or call us.

*CP/M is a trademark of Digital Research.

Structured Systems

5204 Claremont Oakland, Ca. 94618 (415) 547-1567

Circle 115 on inquiry card.

August 1980 © BYTE Publications Inc 169

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.

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 1), 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 <BUILDS 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

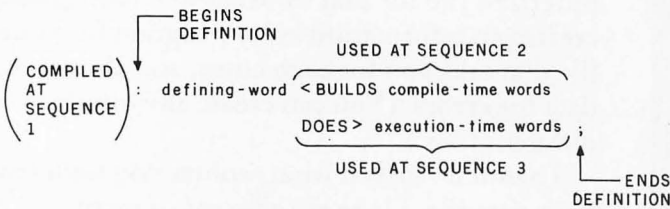


Figure 4: The structure of the source definition of a defining word. These source lines create a defining word for a new family (sequence 1). Execution of the defining word (sequence 2) <BUILDS a dictionary definition for a new family member. The contents of that definition is constructed by the compile-time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

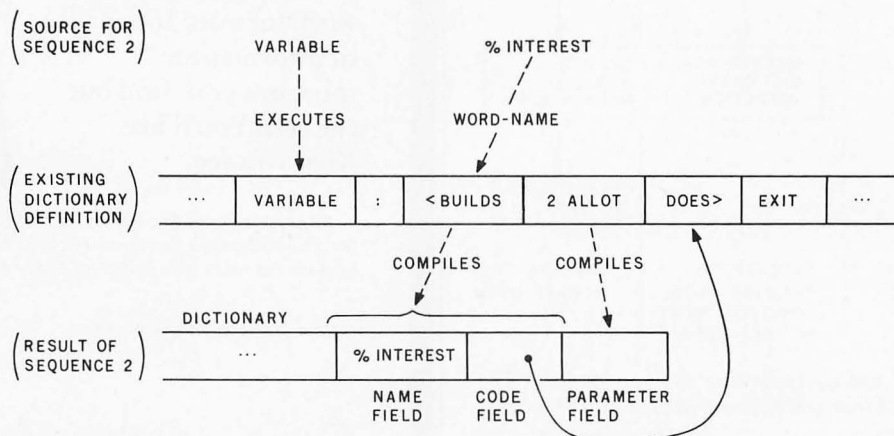


Figure 5: The result of executing a defining word. The first line is executed, resulting in the compilation of the word-name %INTEREST . The result is a new definition in the dictionary.

(%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 re-

quires 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

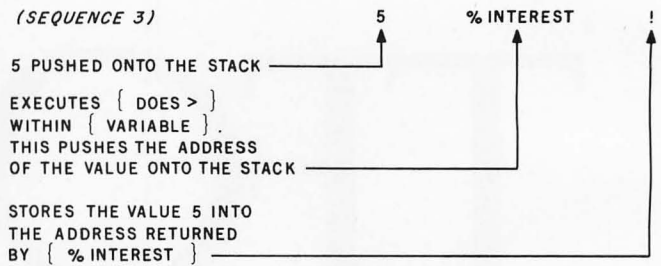


Figure 6: The execution of a family member word. The value 5 is stored in the variable %INTEREST .

64K MEMORY FOR THE HEATHKIT H8* COMPUTER

Assembled	Kit	
\$750	\$650	64K (56K)
615	525	48K
480	400	32K
345	275	16K

Memory Expansion Kit - 16K \$125

PC Board Only - With Documentation \$ 50

Phone for Free Brochure 714/830-2092

*HEATHKIT and H8 are Registered Trademarks of the Heath Co.

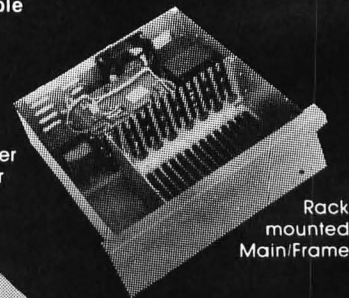


— TRIONYX ELECTRONICS

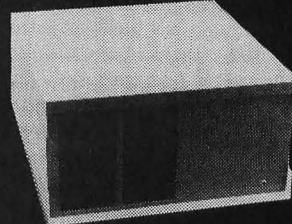
BOX 5131-C, SANTA ANA, CA 92704

Main/Frames from \$200

- 14 Basic Models Available
- Assembled & Tested
- Power Supply:
8v@15A, ± 16v@3A
- 15 Slot Motherboard
(connectors optional)
- Card cage & guides
- Fan, line cord, fuse, power
& reset switches, EMI filter
- 8v@30A, ± 16v@10A
option on some models



Rack mounted Main/Frame



8" Floppy Main/Frame (includes power for drives and main/frames)

Write or call for our brochure which includes our application note:

'Building Cheap Computers' INTEGRAND

8474 Ave. 296 • Visalia, CA 93277 • (209) 733-9288
We accept BankAmericard/Visa and MasterCharge

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

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

- BUDGET:** The heart of a comprehensive home finance system. Allows user to define up to 20 budget items. Actual expense input can be by keyboard or by automatic reading of CHECKBOOK II files. Costs are automatically sorted and compared with budget. BUDGET produces both monthly actual/budget/variance report and a year-to-date by month summary of actual costs. Color graphics display of expenses. . . \$24.95
- CHECKBOOK II:** This extensive program keeps complete records of each check/deposit. Unique check entry system allows user to set up common check purpose and recipient categories. Upon entry you select from this pre-defined menu to minimize keying in a lot of data. Unique names can also be stored for completeness. Rapid access to check files. Check register display scrolls for ease of review. 40 column print-out. Up to 100 checks per month storage. Files accessible by BUDGET program \$19.95
- SAVINGS:** Allows user to keep track of deposits/withdrawals for up to 10 savings accounts. Complete records shown via screen or 40 column printer. \$14.95
- CREDIT CARD:** Keep control of your cards with this program. Organizes, stores and displays purchases, payments and service charges. Screen or 40 column printer display. Up to 10 separate cards \$14.95

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

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

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

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

- MICROACCOUNTANT:** The ideal system for the small cash business. Based on classic T-accounts and double-entry bookkeeping, this efficient program records and produces reports on account balances, general ledger journals, revenue and expenses. Screen or 40 column printer reports. Handles up to 500 journal entries per period, up to 100 accounts. Instructions include a short primer in Financial Accounting. \$49.95
- UNIVERSAL BUSINESS MACHINE:** This program is designed to SIMPLIFY and SAVE TIME for the serious businessman who must periodically Analyze, Plan and Estimate. The program was created using our Universal Computing Machine and it is programmed to provide the following planning and forecasting tools.
CASH FLOW ANALYSIS PROFORMA BALANCE SHEET SOURCE AND USE OF FUNDS
PROFORMA PROFIT & LOSS SALES FORECASTER JOB COST ESTIMATOR
Price, including documentation and a copy of the base program. Universal Computing Machine. \$89.95
- INVOICE:** Throw away your pens. Use the ELECTRONIC INVOICE facsimile displayed on your CRT. The program prompts and you fill in the data. Includes 3 address fields (yours, Bill to and Ship to), Invoice No., Account No., Order No., Salesman, Terms, Ship Code, FOB Pt. and Date. Up to 10 items per sheet with these descriptions: Item No., No. of units, Unit Price, Product Code, Product Description, Total Dollar amount per item and invoice total dollar amount. Generates, at your option, hard copy invoices, shipping memos, mailing labels, audit copies and disc updates to master A/R files. (48K) \$49.95
- BUSINESS CHECK REGISTER:** Expanded version of the Checkbook II program. Handles up to 500 checks per month with complete record keeping. (48K) \$29.95
- BUSINESS BUDGET:** As described above and companion program to Business Check Register. Handles 500 transactions per month, up to 20 cost categories. Accesses BCR files for actual costs. (48K) \$29.95

ELECTRICAL ENGINEERING SERIES: Both programs \$159.95 Apple


- LOGIC SIMULATOR: SAVE TIME AND MONEY.** Simulate your digital logic circuits before you build them. CMOS, TTL, or whatever, if it's digital logic, this program can handle it. The program is an interactive, menu driven, full-fledged logic simulator capable of simulating the bit-time by bit-time response of a logic network to user-specified input patterns. It will handle up to 1000 gates, including NANDS, NORs, INVERTERS, FLIP-FLOPS, SHIFT REGISTERS, COUNTERS and user-defined MACROS. Up to 40 user-defined, random, or binary input patterns. Simulation results displayed on CRT or printer. Accepts network descriptions from keyboard or from LOGIC DESIGNER for simulation. Specify 1000 gate version (48K required) or 500 gate version (32K required) \$89.95
- LOGIC DESIGNER:** Interactive HI-RES Graphics program for designing digital logic systems. A menu driven series of keyboard commands allows you to draw directly on the screen up to 15 different gate types, including 10 gate shape patterns supplied with the program and 5 reserved for user-specification. Standard patterns supplied are NAND, NOR, INVERTER, EX-OR, T-FLOP, JK-FLOP, D-FLOP, RS-FLOP, 4 Bit COUNTER and N-BIT SHIFT REGISTER. User interconnects gates just as you would normally draw using line graphics commands. Network descriptions for LOGIC SIMULATOR generated simultaneously with the CRT diagram being drawn. Drawing is done in pages of up to 20 gates. Up to 50 pages (10 per disc) can be drawn, saved and recalled. Specify 1000 gate (48K) or 500 gate (32K) system \$89.95

MATHEMATICS SERIES: Complete Package \$49.95 Apple only

- NUMERICAL ANALYSIS:** HI-RES 2-Dimensional plot of any function. Automatic scaling. At your option, the program will plot the function, plot the INTEGRAL, plot the DERIVATIVE, determine the ROOTS, find the MAXIMA and MINIMA and list the INTEGRAL VALUE. For 16K \$19.95
- MATRIX:** A general purpose, menu driven program for determining the INVERSE and DETERMINANT of any matrix, as well as the SOLUTION to any set of SIMULTANEOUS LINEAR EQUATIONS. Disk I/O for data save. Specify 55 eqn. set (48K) or 35 eqn. (32K) \$19.95
- 3-D SURFACE PLOTTER:** Explore the ELEGANCE and BEAUTY of MATHEMATICS by creating HI-RES PLOTS of 3-dimensional surfaces from any 3-variable equation. Disc save and recall routines for plots. Menu driven to vary surface parameters. Demos include BLACK HOLE gravitational curvature equations. \$19.95

ACTION ADVENTURE GAMES SERIES: Entire series \$29.95 Apple only

- RED BARON:** Can you outfly the RED BARON? This fast action game simulates a machine-gun DOG-FIGHT between your WORLD WAR I BI-PLANE and the baron's. You can LOOP, DIVE, BANK or CLIMB in any one of 8 directions - and so can the BARON. In HI-RES graphics \$14.95
- BATTLE OF MIDWAY:** You are in command of the U.S.S. HORNETS' DIVE-BOMBER squadron. Your targets are the Aircraft carriers, Akagi, Soryu and Kaga. You must fly your way through ZEROS and AA FIRE to make your DIVE-BOMB run. In HI-RES graphics \$14.95
- SUB ATTACK:** It's April, 1943. The enemy convoy is headed for the CORAL SEA. Your sub, the MORAY, has just sighted the CARRIERS and BATTLESHIPS. Easy pickings. But watch out for the DESTROYERS - they're fast and deadly. In HI-RES graphics \$14.95
- FREE CATALOG-**All programs are supplied in disc and run on Apple II w/Disc & Applesoft ROM Card & TRS-80 Level II and require 32K RAM unless otherwise noted. Detailed instructions included. Orders shipped within 3 days. Card users include card number. Add \$1.50 postage and handling with each order. California residents add 6% sales tax. Make checks payable to:

 **SPECTRUM SOFTWARE**
DEALER INQUIRIES P.O. BOX 2084 - 142 CARLOW, SUNNYVALE, CA 94087
INVENTED FOR PHONE ORDERS - 408-738-4387

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.

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


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

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.)

```

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

: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

```

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.

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

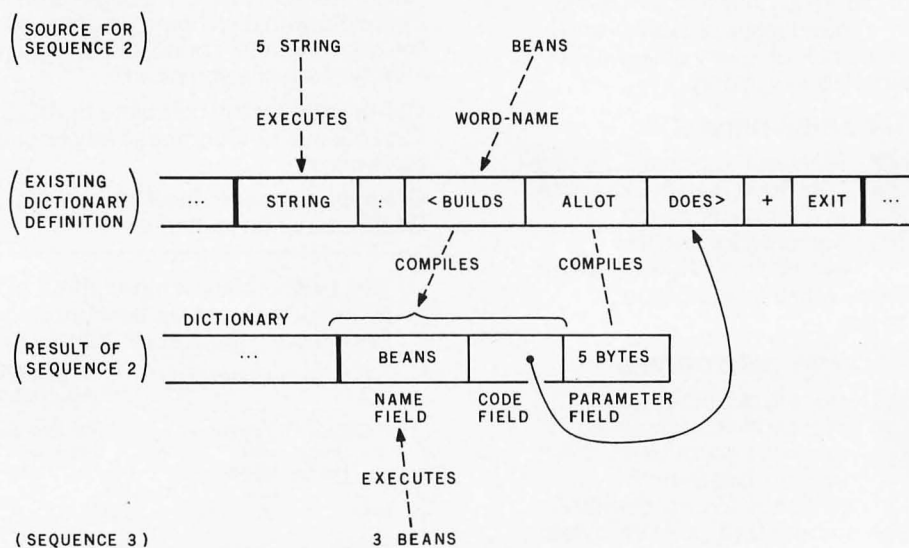


Figure 7: The creation and use of a character array. The defining word STRING is executed, causing the compilation of a dictionary definition for BEANS containing 5 bytes of data space. When BEANS is executed (last line), the DOES> part of the definition of STRING adds the address of the parameter field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

CP/M® SOFTWARE TOOLS NEW ED-80 TEXT EDITOR

ED-80 offers a refreshing new approach for the creation and editing of program and data files conversationally—and it saves you money. Its powerful editing capabilities will satisfy the most demanding professional—yet it can still be used by the inexperienced beginner.

Look at These Outstanding Features:

- FULL SCREEN window displays with forward and backward scrolling for editing your data a page-at-a-time, rather than line-by-line.
- Provides you with all the features found on the large mainframe and minicomputer editors, such as IBM, UNIVAC, CDC, and DEC.
- Commands include forward or backward LOCATE, CHANGE, and FIND; and INSERT, DELETE, REPLACE, APPEND, SAVE, PRINT, WINDOW, MACRO, TABSET, SCALE, DUMP, and others.
- Compatible with existing CP/M edit and text formatted files, with CBASIC, and with Microsoft's MBASIC, FORTRAN, COBOL, and ASSEMBLER.
- CHANGE commands allow you to make conditional changes and to use variable length strings.
- Designed for CP/M and derivative operating systems, including LIFEBOAT, CDOS, IMDOS, DOS-A, ADOS, etc.
- GET and PUT commands for concatenating, moving, duplicating, and merging your edit files on the same or different diskettes.
- Provides you with fast memory-to-memory COPY commands, and an intermediate buffer for copying lines over-and-over.
- Saves your last LOCATE, CHANGE, FIND, and APPEND command for easy re-execution.
- Simple line-oriented commands for character string editing.
- Safeguards to prevent catastrophic user errors that result in the loss of your edit file.
- INLINE command for your character-oriented editing.
- Designed for today's CRT's, video monitors, and teletypewriter terminals.
- Thoroughly field tested and documented with a comprehensive User's Manual and self-instructional tutorial.

And remember — in today's interactive programming environment — your most important software tool is your text editor. ED-80 is already working in industry, government, universities, and in personal computing to significantly cut program development time and to reduce high labor costs. Why not let ED-80 begin solving your text editing problems today? ED-80 is protected by copyright and furnished under a paid-up license for use on a single computer system. Single Density Diskette and Manual: \$99.00, or the Manual alone: \$20.00 (credited with purchase of the Diskette). Specify Disk make/model, 5" or 8", hard or soft sectored. ORDER NOW and we'll pay the postage!

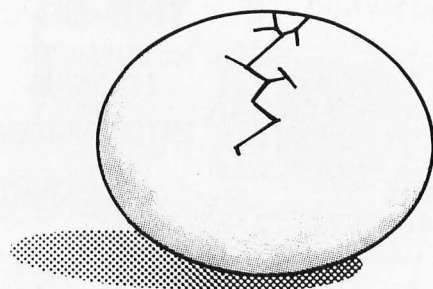
SOFTWARE DEVELOPMENT & TRAINING, INC.

Post Office Box 4511 — Huntsville, Alabama 35802

Dealer Inquiries Welcomed

© CP/M is a trademark of Digital Research

The 2nd Generation... It's all that it's Cracked up to be.



MEASUREMENT
systems & controls
incorporated

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:

$$\text{ARRAY}(5,7,2) = \text{AR1}(1,2) + \text{AR2}(10,20,30)$$

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

Marymac Industries Inc

Radio Shack®

AUTHORIZED SALES CENTER

Save 10% 15%

OR MORE

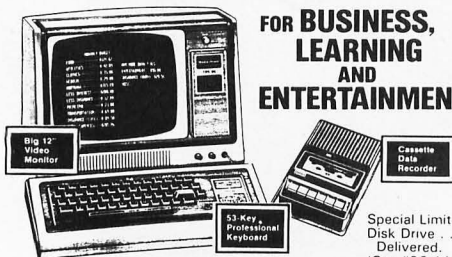
DELIVERED TO YOUR DOOR

Owned and operated by Marymac Industries Inc. Houston's only independent Radio Shack® dealer. Warranties will be honored by all company owned Radio Shack® stores and most franchise and dealer authorized sales centers. Store open Mon.-Sat. 10-7. We pay freight and insurance. Save state sales tax. Texas residents add only 5% sales tax. Brand new in factory sealed cartons. Reference: Katy National Bank. Call us for a customer reference near your city. Offered exclusively by Radio Shack® Authorized Sales Center 21969 Katy Fwy., Katy (Houston) Texas 77450

Telephone 1-713-392-0747

TRS-80™

FOR BUSINESS,
LEARNING
AND
ENTERTAINMENT



Special Limited Time Only
Disk Drive . . . \$424.90
Delivered.
(Cat.#26-1160, 26-1161)

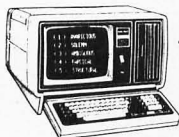
Meet TRS-80's Big Brother!

The New TRS-80 Model II

We are located just 5 hours from the giant Tandy Computerware House in Ft. Worth, Texas.

Call
Joe McManus
Today

We've added a bigger, more powerful "brother" to the TRS-80 family. Its TRS-80 Model II — a completely new microcomputer for business applications.



CHARGE IT



GENERAL LEDGER PAYROLL ACCOUNTS RECEIVABLE & PAYABLE

Flexible and sophisticated business software that is among the highest quality on the market. Originally developed by OSBORNE & ASSOCIATES and rapidly becoming a standard. Our service is support. We will send you these programs with the proper I/O and CRT specific subroutines for your hardware configuration. Get back to business and leave the programming to us. Include hardware description with order.

- Accounts Receivable and Payable 145.00
- Payroll (California) 145.00
- Non California state tax calculations (please inquire) 15-250.00
- General Ledger 145.00
- Multiple profit center option for G/L 25.00
- Manuals (each) 20.00

All programs in CBASIC under CP/M (includes source)

These programs are up and running on the following computer systems: Altos, TRS-80 MOD II (under CP/M), Northstar, Vector Graphics, Intertec Super Brain, Cromemco, and others.

Synergetic Computer Products

508 University Ave • Palo Alto, CA 94301
(415) 328-5391

• Visa • Mastercharge • COD • Certified Check
CP/M is a trademark of Digital Research

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 )

```

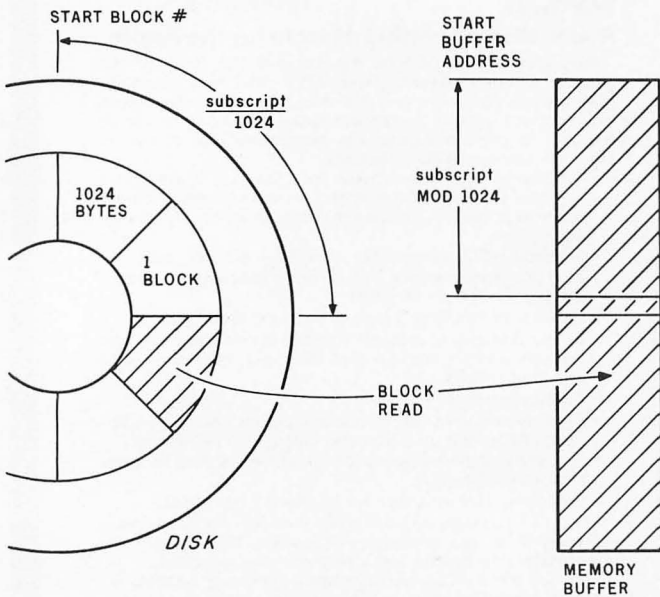


Figure 8: Accessing a virtual array. The data for a large array is kept on a disk. When a byte is referenced, BEANS is executed. One block containing the byte is read into a memory buffer (if it is not already present). Finally, the memory address of the referenced byte is returned by BEANS .

WE DELIVER!

Osborne Business Software



- Ready to run — no recompiling!
- Custom configured for your terminal.
- We are committed to fully supporting our users.
- One year maintenance included in price.
- Source programs, with enhancements.

General Ledger with Cash Journal	\$95
Accounts Payable	\$95
Accounts Receivable	\$95
Payroll with Cost Accounting	\$95
All four packages	\$295

Formats: 8", NorthStar, TRS-80 MOD II im. Manuals are not included in the above prices — add \$20 per manual desired (AR/AP are in one manual). CP/M and CBASIC2 required. Users must sign licensing agreement. Dealer inquiries invited.

Other high-quality CP/M software available — contact us for our complete price list. Some examples:

WORDSTAR	\$435	TEXTWRITER III	\$120
PEARL II	\$345	PEARL III	\$645
PASCAL/Z	\$385	TINY-C	\$ 95
CP/M and CBASIC2 for TRS-80 MOD II im (PGT)			\$285

To order call: (206) 542-8370
or write: **VANDATA**
17541 Stone Avenue North
Seattle, WA 98133

VISA/MC/COD Welcome — TRS-80 is a registered — of Radio Shack Inc.

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: } .

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

```
( sequence 2 )   CASE: ANIMAL  OPET 1PET 2PET ;
(b)
```

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

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 )
```

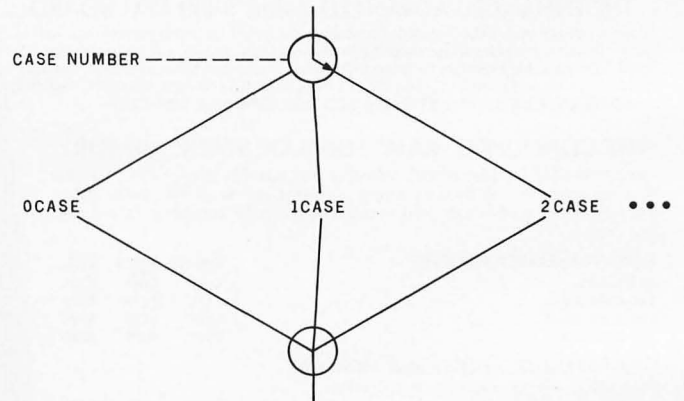


Figure 9: The function of a case control structure. The case number selects one of several procedures for execution, then continues along a single exit path.

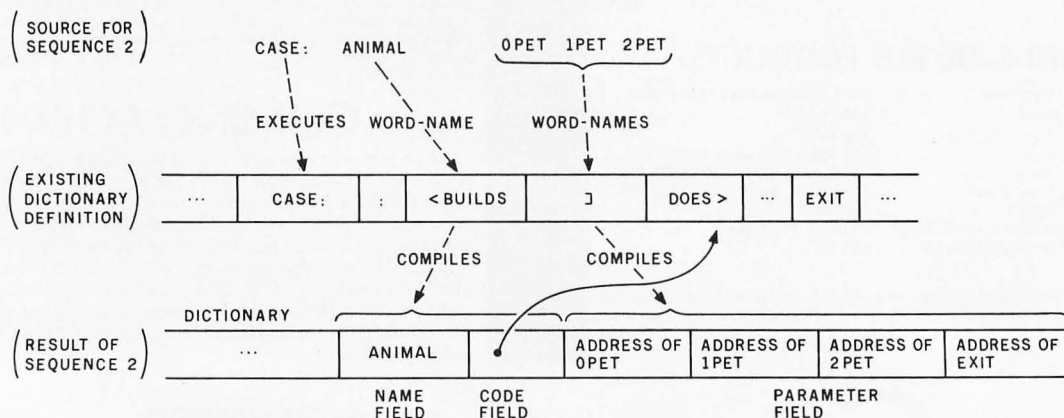


Figure 10: The creation of a case control word. The execution of { CASE: } causes a definition for ANIMAL to be appended to the dictionary. The ']' word uses the { : } compiler to compile the addresses of the case words following ANIMAL .

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 ]
      DOES > 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. ■

NOBODY DOES IT LIKE SYNCHRO-SOUND!



HAZELTINE 1420 VIDEO TERMINAL

\$775⁰⁰



HAZELTINE 1500 VIDEO TERMINAL

\$849⁹⁵



SYNCHRO-SOUND
The Computer People
193-25 Jamaica Avenue
Jamaica, N.Y. 11423

ENTERPRISES, INC.

PHONE ORDERS. CALL:
New York—212/468-7067
Los Angeles—213/628-1808
Chicago—312/641-3010
Dallas—214/742-6090

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 <i>follows</i> 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.)

/	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

TARGET HOST ► TARGET HOST ► TARGET HOST

CROSS COMPILE FORTH!

CROSS COMPILING IS THE MOST CONVENIENT WAY TO IMPLEMENT AND EXTEND FORTH. NOW YOU CAN CROSS COMPILE AN ENTIRE FORTH SYSTEM WITH ALL FORWARD REFERENCES RESOLVED IN A SINGLE PASS TO PRODUCE AN EXECUTABLE IMAGE IN MEMORY OR ON DISK AND A LOAD MAP OF ALL DEFINED SYMBOLS. THE CROSS COMPILER IS WRITTEN IN HIGH LEVEL FORTH INTEREST GROUP (FIG) FORTH. A COMPLETE DESCRIPTION OF EACH WORD IN THE CROSS COMPILER IS GIVEN WITH STEP BY STEP STACK CONTENTS. FORTH INTERNALS (NEXT, BUILD, DOES, CREATE, ETC.) ARE ALSO COMPLETELY DESCRIBED. A CROSS COMPILABLE VERSION OF THE FIG MODEL 1.0 IS PROVIDED FOR THE 8080 WITH AN ASSEMBLER / DISASSEMBLER. THIS MAY BE EASILY CONVERTED TO ANY MACHINE. A DETAILED DESCRIPTION IS GIVEN FOR FIRST TIME IMPLEMENTATIONS. THE ENTIRE PACKAGE IS AVAILABLE FOR \$70. FROM:

Nautilus Systems
P.O. Box 1098
Santa Cruz, CA. 95061

FOR THE SERIOUS FORTH USER

TARGET HOST ► TARGET HOST ► TARGET HOST

ANNOUNCING:

NEW!

MICROSTAT

A complete statistics package for business, scientific, education and research work. No other package has the features of MICROSTAT. For example:

- File oriented with COMPLETE editing
- A Data Management Subsystem for editing, sorting, ranking, lagging, data file transfers PLUS 11 data transformations (e.g., linear, reciprocal, exponential, etc.)
- Frequency distributions
- Simple and multiple regression
- Time series (including exponential smoothing)
- 11 Non-parametric tests
- Crosstabs/Chi-square
- Factorials (up to 1,000,000!), permutations, combinations
- 8 Probability distributions
- Scatterplots
- Hypothesis test (Mean, proportion)
- ANOVA (one and two-way)
- Correlation
- Plus many other unique features

Users manual: \$10.00 (credited towards purchase) and includes sample data and printouts. Uses

NORTH STAR BASIC 32K of memory, one or two disk drives (2 recommended). Printer optional. Price: \$200.00



ECOSOFT

P.O. Box 68602
Indianapolis, IN 46268

Phone orders:
(317) 253-6828

{ <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) <i>during the definition of a word.</i>
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.

SURPLUS "SELECTRIC" SPECIAL!

"SELECTRIC" TYPEWRITER TERMINAL

Just imagine; an IBM Model 725 "SELECTRIC" typewriter built into a complete table-top RS-232 terminal! These surplus terminals were formerly on lease and appear to be in good condition (we test 'em to make sure the printer is functional!) These fantastic BCD-Coded terminals feature:

<ul style="list-style-type: none"> • 15" CARRIAGE • 725 'SELECTRIC' • RS-232 I/O • 132 COLUMNS • Sim. to IBM 2741 • Std. Typewriter Kbd. • MAX: 15 CPS RATE • 10 Chars./Inch • Removeable Type Sphere 	<ul style="list-style-type: none"> • 134.5 BAUD I/O • 88 Character Set • 6 Bit BCD CODE • Attractive Case • Upper/Lower SHIFT
--	--



ONLY
\$469⁰⁰!
Ea. ■

While we will check out each unit, we MUST offer these unique bargains "AS-IS": Meaning they may need some service but are basically operational. Add \$20.00 for packing crate, you pay shipping on delivery.

ALSO INCLUDES: Type ball, I/O circuit boards, power supply & some data. Sorry, no power cord included.

—SPECIAL OFFER!—

Buy 2, take 20% Off the Full Price—
You Pay Only **2 for \$750⁰⁰**

"SELECTRIC" PRINTER MAINTAINANCE MANUAL
JUST IN!! We now have available some excellent printer maintenance manuals. These are the most thorough manuals we've seen. Well worth the price! ONLY **\$25.00**ea.
* "SELECTRIC" is an IBM Trademark

CFR Associates, Inc.

MAIL ADDRESS: P.O. Box 144 NEWTON, N.H. 03858	WAREHOUSE: 18 GRANITE STREET HAVERHILL, MASS. 01830	(617)372-8536 <small>Phone Orders Are Welcome</small>
---	---	--

At Last! HIGH RESOLUTION S-100 GRAPHICS



Unretouched photograph

<p>512 x 640 DOT RESOLUTION S-100 PLUG IN COMPLETE INTERFACE ON-BOARD MEMORY ASSEMBLED & TESTED FROM \$1,200</p>	<p>OPTIONS:</p> <ul style="list-style-type: none"> • 16 COLORS • GREY LEVELS • LIGHT PEN • SOFTWARE
---	--

Send for brochure and data



CAMBRIDGE DEVELOPMENT LAB
44 Brattle Street Cambridge, MA 02138

{ 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

MICRO MISCELLANY	
APPLE II PARALLEL INTERFACE  \$79.95 Interfaces printers, synthesizers keyboards, and JBE A-D D-A Converter & Switches. This interface has 4 I/O ports with handshaking logic, 2-6522 VIA's and a 74LS74 for timing. Inputs and outputs are TTL compatible. 79-295K Complete Kit \$69.95 79-295A Assembled \$79.95	SOLID STATE SWITCH  \$44.95  \$12.50 Your computer can control power (120VAC) to your printer, lights, and other 120VAC appliances up to 720 watts (6AMPS at 120VAC). Input 3 to 15 VDC, 2-13 MA TTL compatible, isolation 1500V. 79-282 1 Channel Kit \$ 9.95 Assm. \$12.50 79-282 4 Channel Kit \$34.95 Assm. \$44.95
AtoD DtoA CONVERTER  \$69.95 Analog to Digital, Digital to Analog Converter, AtoD conversion time 20us. DtoA conversion 5us. Uses include speech and music synthesizing and slow scan TV. Single power supply (5V), 8 Bits wide, latched I/O, strobe lines. 79-287K Complete Kit \$49.95 79-287A Assembled \$69.95	BARE BOARDS SINGLE BOARD COMPUTERS 8088 5-CHIP SYSTEM \$29.95 8085 3-CHIP SYSTEM \$24.95 MEMORY BOARD 8208 64K DYNAMIC \$39.95 ALL PRODUCTS AVAILABLE FROM: JOHN BELL ENGINEERING P.O. Box 338 Dept. 4 Redwood City, CA 94064 (415) 367-1137 Add 6% sales tax in California and \$1.00 shipping and handling for orders less than \$20. Add 4% for VISA or M.C.
JOHN BELL ENGINEERING	

CAT-100 FULL COLOR GRAPHICS

The original 256-color imaging system with high resolution video FRAME GRABBER for the S-100 bus.

Capture and digitize a video frame in 1/60 of a second. Select the best resolution for your application, from 256 to 1280 pixels per TV line. Display your digitized or computer processed image with 256 gray levels or 256 colors on standard B&W, NTSC or RGB color TV monitors.



Compact two-board basic system

Features:

- Highest possible quality 480x512x8 digital video image presently available on the market
- Input capability from TV camera or other sources
- Variety of synchronization choices
- 2 selectable video A/D conversion circuits
- Choice of 1, 2, 4, 8, 16 or 32 bits per pixel
- 32K-byte image memory on the basic system
- 32, 64, 128 & 256K byte system capacity
- Lightpen input
- Photographic trigger control input
- Software selectable system parameters
- Interfaces for TRS-80 and other processors
- Comprehensive line of accessories, monitors and support software

SEND FOR FREE CATALOG



240x256 Digitized image, 16 levels



480x512 Computer-generated

DIGITAL GRAPHIC SYSTEMS
 441 California Ave., Palo Alto, CA 94306 415/494-6088

COLOR SOFTWARE

Unless otherwise noted all programs are \$15 each, for Apple II, Atari 16K, TI 99/4

UNITS: Practice converting yards-feet-inches, pounds-ounces, metric units, etc.

3-D STARTREK: Discover new planets, fight Klingons in 3-dimensional galaxy.

ROADRACE: Race around 2.25 mile course. 1 or 2 players. Not for TI 99/4.

FRACTIONS: Practice adding, subtracting, multiplying and comparing fractions.

MAJOR LEAGUE BASEBALL: Manage Major League teams and make all lineup, batting, pitching and running decisions. \$25. Apple II with 48K, Applesoft ROM and one disk.

BLACKJACK: Popular card game for 1 to 3 players. Not for Apple II.

NUCLEAR REACTOR: Realistic dynamic model of nuclear power plant in operation.

COLOR SOFTWARE, 5410 W. 20th St., Indianapolis, IN 46224

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.

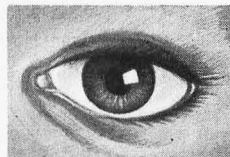
IN STOCK

A/D - D/A

S-100 A/D
8 Ch. Differential or
16 Ch. Single-Ended,
12 Bit, High Speed \$495.

S-100 D/A 4 Channel
12 Bit, High Speed \$395.

TRS-80 A/D-D/A
12-Bit, High Speed
Available Soon



S-100 VIDEO DIGITIZATION

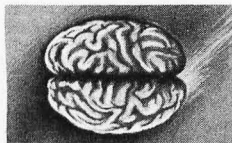
Real Time Video \$850.
Digitizer and Display
Computer Portrait
System \$4950.

S-100 Boards

Video and/or Analog
Data Acquisition
Microcomputer Systems



The High Performance S-100 People
TECMAR, INC.
23414 Greenlawn • Cleveland, OH 44122
(216) 382-7599



S-100 8086

CPU with \$450.
Vectored Interrupts
PROM-I/O \$495.
RAM \$395.
8K x 16/16K x 8
Parallel I/O \$350.
and Timer

H	0	1	2-byte variable containing address of the top of the dictionary; execution of this word causes the <i>address</i> 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 <i>value</i> 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 <i>address</i> 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 <i>lower 8 bits</i> 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 <i>address</i> 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. ■