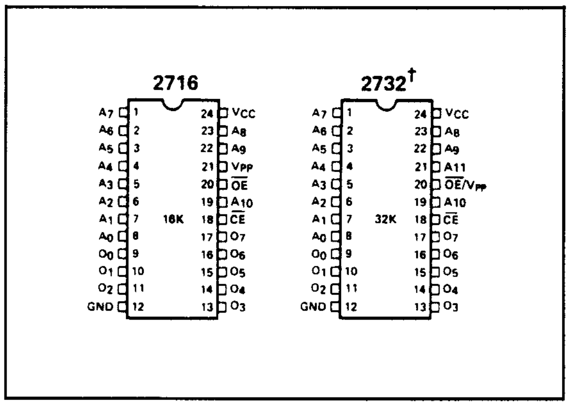
# FORTH DIMENSIONS

VOLUME IV, NUMBER 4 $2.50

## INSIDE:



2716
16K
A7 A6 A5 A4 A3 A2 A1 A0 O0 O1 O2 GND
VCC A8 A9 VPP OE A10 CE O7 O6 O5 O4 O3
2732†
32K
A7 A6 A5 A4 A3 A2 A1 A0 O0 O1 O2 GND
VCC A8 A9 A11 OE/VPP A10 CE O7 O6 O5 O4 O3

## CODING FOR ROM

# FORTH PROGRAMMING AIDS

**from Curry Associates**

***FORTH PROGRAMMING AIDS*** is a software package containing high-level FORTH routines that allow you to write more efficient programs in less development time. It is also useful for maintaining existing FORTH programs. The ***FPA*** package includes four modules:

**TRANSLATOR** provides a one-to-one translation of FORTH run-time code.

**CALLFINDER** finds calling words, i.e. calls *to* a specific word.

**DECOMPILER** generates structured FORTH source code from RAM and inserts program control words (*e.g.,* IF, ELSE).

**SUBROUTINE DECOMPILER** finds called words, *i.e.,* words called *by* a specific word, to all nesting levels.

***FORTH PROGRAMMING AIDS*** enables you to:

- Minimize memory requirements for target systems by finding only those words used in the target application.
- Tailor existing words (including nucleus words) to specific needs by decompiling the word to disk, editing, and recompiling.
- Build on previous work by extracting debugged FORTH routines (including constants and variables) from RAM to disk.
- Patch changes into existing compiled words in seconds.

***FORTH PROGRAMMING AIDS*** comes with complete source code and a 50-page, indexed manual.

**The DECOMPILER alone is worth a second look.** This is a *true* decompiler which converts the FORTH words in RAM into compilable, structured FORTH source code, including program control words such as IF, ELSE, THEN, BEGIN, etc. If you ask ***FPA*** to DECOMPILE the nucleus word INTERPRET, you get the following output displayed on your terminal within 3 seconds:

```
            ( NFA&PFA: 4796 4810 )
  : INTERPRET
      BEGIN -FIND
          IF STATE @ <
             IF CFA .
             ELSE CFA EXECUTE
             THEN ?STACK
          ELSE HERE NUMBER DPL @ 1+
             IF [COMPILE] DLITERAL
             ELSE DROP [COMPILE] LITERAL
             THEN ?STACK
          THEN
      AGAIN ;
```

You can decompile one word, or a range of words at one time — even the whole FORTH system! This decompiled output may be sent by ***FPA*** options to the console, printer, or disk.

DECOMPILE is useful for looking up words, or for obtaining variations of words by decompiling to disk, editing, and recompiling.

**System Requirements:** FORTH nucleus based on the fig-FORTH model or 79-STANDARD; a minimum of 3K bytes and a recommended 13K bytes of free dictionary space.

For more information, call **Ren Curry 415/322-1463** or **Tom Wempe 408/378-2811**

Yes, send me a copy of ***FORTH PROGRAMMING AIDS***, including all source code and the 50-page manual.

| | | |
|---|---|---|
| ☐ fig-FORTH model | $150 | Calif. residents add 6.5% tax. |
| ☐ FORTH-79 STANDARD (specify system) | $150 | Foreign air shipments add $15. |
| ☐ Manual alone (credit toward program purchase) | $25 | |
| ☐ Send more information | | |

☐ Master Charge ☐ Visa Account Number ______________ Exp. Date ______________

Name ______________________________

Company ______________________________

Street ______________________________

City/State/Zip ______________________________

Indicate disk format:

☐ 8" ss/sd fig-FORTH screens
☐ 8" ss/sd CP/M™ 2.2 file
☐ Apple 3.3
☐ PC FORTH
☐ Other ______________

Send to: **Curry Associates, P. O. Box 11324, Palo Alto, CA 94306 415/322-1463 or 408/378-2811**

# Letters . . .

## 2CONSTANT and/or DCONSTANT

Dear FIG,

Charles H. Moore recently explained to me the origin of **2CONSTANT**, **2VARIABLE**, **2DUP** etc. In spite of my earlier opinion, that we certainly do not mean 2 constants with **2CONSTANT**, he really did mean 2 constants. The entire family of words were originally used to apply to two integers! The two intergers were ratios to be used with operations such as ***/**. For example:

```
355 113 2CONSTANT PI
```

Then to multiply a value by PI:

```
1000 PI */
```

This all makes sense to me now. See the footnote on page 122 of *Starting* FORTH for a handy table of similar ratios.

The implementation of these words usually is the same as the implementation necessary for operations on double precision integers. Thus it might be argued that there is no reason for different names for the double number word set.

However, Charles Moore also pointed out to me that the implementations are not always the same. I gather that they are never the same in 32-bit CPUs.

We have a precedent for different names with the same function — for example: **R@** and **I**. Though the implementation may be the same for **2CONSTANT**, with two single integer constants, and **DCONSTANT**, for a double precision constant, the meaning is different. As long as the implementation remains the same there is no conflict in using one as an alias for the other. But beware!

*Glen B. Haydon*
*La Honda, California*

## Neater S–>D

Dear FIG,

In Vol. IV, No. 1 Robert Smith gave us a definition of **S–>D** to convert 16 to 32 bit integer. We have used a much neater one for a long time:

```
: S->D   DUP 0< MINUS ;
```

Think about it!

*J.R. (Roger) Stapleton*
*University Observatory*
*St. Andrews, Fife.*
*Scotland*

## FORTH Machine Project

Dear FIG,

I would like to extend an invitation to anyone interested to take part in a volunteer project to build a FORTH machine. Some support will be coming from Advanced Micro Devices, including one of their new 29116 16 bit wide microprogrammable chips. Anyone interested in this should contact me at the address and phone number below.

*Martin Schaaf*
*1100 N. Placentia, #E-38*
*Fullerton, California 92630*
*(714) 993-7128*

## Reflections

Dear FIG,

It is said that introspection is good for the soul. Let's slow down and reflect for a few moments on the marvelous invention wich we know as FORTH and the industry which has sprung up around it.

It has become clear that Charles Moore's efforts to increase his productivity as an applications programmer has had and will continue to have a substantial impact on the software industry. What has caused this remarkable phenomenon?

— FORTH was designed and implemented by an eminently capable committee of 1.
— FORTH was hammered into its current form on the hard anvil of actual applications experience.
— FORTH was the by-product of work done under a government grant, rather than the object of the grant.
— A small group of applications programmers who had experienced the awesome power of FORTH recognized the critical lack of FORTH vendors and selflessly conspired to develop and place in the public domain FORTH implementations for all of the major microprocessors. Thus came the birth of FIG and a revolution in the software industry.

These efforts have resulted in a burgeoning FORTH community complete with a deluge of vendors offering a wide range of products, a broadly based interest group, and a lengthy list of successful projects in which FORTH has played a major role. One can almost feel the ground rumble as IBM, HP, GE, and other industry giants start to sit up and take notice.

Although this all sounds so wonderful, I have some grave concerns over the state of our young industry.

First, we have failed to publicly recognize the individual contributions of our teachers. Although we stand on the shoulders of giants, we have systematically failed to recognize them for the tremendous contributions they have made.

Who publicly acknowledges the debt to Charles Moore? or Bill Ragsdale? What recognition has Dean Sanderson received? How strongly do vendors of FIG model derivatives acknowledge their debt to the implementors? What mechanism exists to express our gratitude to these people?

Perhaps, due to the effort required to learn the internals of FORTH, we overemphasize our own intellect at the expense of our teachers. No other successful philosophy or technique which I am aware of has this ominous attribute.

*Letters continued on next page*

**FORTH Dimensions**
Published by FORTH Interest Group
Volume IV, No. 4
November/December 1982
Editorial/Production
Leo Brodie
Publisher
Roy C. Martens

FORTH Dimensions solicits editorial material, comments and letters. No responsibility is assumed for accuracy of material submitted. Unless noted otherwise, material published by the FORTH Interest Group is in the public domain. Such material may be reproduced with credit given to the author and the FORTH Interest Group.

Subscription to FORTH Dimensions is free with membership in the FORTH Interest Group at $15.00 per year ($27.00 foreign air). For membership, change of address and/or to submit material, the address is: FORTH Interest Group, P.O. Box 1105, San Carlos, CA 94070

Second, we have become confused over what FORTH is. We have attempted to force a comprehensive applications programming approach into the confines of what has traditionally been termed a programming language. We are guilty of placing constraints on what FORTH can be and endowing it with features it was never meant to have.

Standard FORTH programs currently must limit themselves to string I/O to the system console and block I/O to the first 32 blocks available on mass storage. Although it is possible to fetch and store data within the 64K byte addressable range, it is unlikely that hardware specific addresses or data formats will be relevant, unless the program is transported to an identical configuration. Given a reasonable amount of mass storage, Data Reduction, DataBase manipulation and self-contained algorthms appear to be the only likely candidates for transportation across standard systems.

The goal of standardization is to allow the transportation of hardware and configuration independent algorithms across standard FORTH systems. Few existing FORTH applications are either hardware or configuration independent. In the past, FORTH has been applied mostly to solving applications which were distinguished by their hardware and configuration dependent nature. Without the ability to extend and adapt itself around such environments, FORTH would not be where it is today.

It has been my experience that portions of applications which control specific hardware are transportable at two levels. Either as a generic algorthm for supporting a specific class of hardware (i.e., graphics) or as a specific algorithm to support a specific device in a new environment (i.e., disc conroller). Both cases typically require some level of tailoring in the new environment.

As a member of the standards team, I am concerned with how to specify a "standard" FORTH environment which is flexible enough to handle both cases and still provide the transportability we all desire.

Third, although we have been extremely successful in providing a tremendously powerful tool to a large number of people, we have failed to teach them how to effectively use it. As a result, a lot of effort has been spent on careful inspection and infatuation with the tool rather than experiencing the benefits of its use.

It is said that the first two applications that a new FORTH programmer writes are a de-compiler and a screen editor (order dependent on whether or not source code for the system was supplied). I would feel more comfortable if the majority of new FORTH programmers were immediately able to use the tool to solve an application problem. The best complement we can bestow upon our teachers is effective use of what they taught us.

Finally, if the FORTH community is to continue to grow, we must all become more business-like. While recognizing that availability of low cost, public domain versions has been instrumental in the proliferation of the language, we are quickly approaching the point when we will have reached all those hardy souls willing to provide enough of their own efforts to overcome the limited documentation and support implications of such low cost versions. We are already seeing FORTH rejected for perfectly feasable applications due to unwillingness on the part of the customer to shoulder this burden.

As is typical of the software industry, we all underestimate the cost and effort involved in developing and supporting a software product. I claim that the total net worth of all FORTH vendors is under 3 million dollars, and that the total yearly revenues resulting directly from FORTH goods and services is under 5 million dollars.

I know of few reasonably profitable FORTH vendors and suspect that most are just meeting expenses. Unfortunately, I don't see this picture changing substantially in the near future.

Although I can conceive of nothing I enjoy more than providing tools and solutions to technically challenging applications, I am concerned that by trading fun for cash, we do ourselves and our industry a disservice. The accumulation of capital would provide the necessary management and support services (i.e., product specific documentation) which I feel is so desperately needed.

You may accuse me of presenting views intended to feather my own bed as a FORTH vendor. I would like to believe that my interests are more highly motivated by the concerns that we more suitably honor our teachers. A healthy industry can better repay this debt.

*Don Colburn*
*FORTH Programmer*

## Closer Approximations

*Editor's Note: Some of you may be interested in this letter I recently received:*

Dear Mr. Brodie,

I am reading and enjoying your book, *Starting FORTH.* I found some better approximations for your footnote on page 122. Here they are, along with the ones from page 122:

| Definition | Approximation | (Approx.-Def.)/Definition | |
|---|---|---|---|
| e | 28667 / 10546 | − 5.7 | x $10^{-9}$ |
| " | 25946 / 9545 | − 2.0 | " |
| $\sqrt{10}$ | 22936 / 7253 | 5.7 | " |
| " | 27379 / 8658 | 0.67 | " |
| Ln2/16.384 | 485 / 11464 | 91. | " |
| " | 846 / 19997 | −12. | " |

*Robert T. Corry*
*Polytechnic University of New York*
*Brooklyn, New York*

## Virginia Figs?

Dear FIG,

Do you have any method of helping members to locate other members? The Potomac FORTH Interest Group is currently the closest I know of, and 120 miles is a long way to go. I would like to see a central Virginia FIG organized, or organize one. Any suggestions?

*John C. Lundin, Jr.*
*Richmond, Virginia*

*John, let's hope your letter stirs up some interest in the Richmond area. See "Start Your Own FIG Chapter" on page 5. —Editor*

*Letters continued on next page*

## Letters . . . (cont.)

### Cleaner Stepper Driver

Dear FIG,

FORTH certainly is a splendid language for controlling stepper motors, and the method shown in the excellent article by Martin Petri and Leo Brodie can be compacted still further by reducing the outputs table to just 4 bytes.

In my implementation, each motor is assigned a byte-variable **(X-PHASE, Y-PHASE)**, whose value is restricted to 0-3 and which is a pointer to the current location of the motor phase within the table sequence 5,6,A,9. The word **STEPCODE** expects on the stack a direction flag and the address of the **_PHASE** word for the motor to be stepped. **STEPCODE** fetches the current pointer, increments or decrements it according to the direction flag, **AND**s the result with 0003 to maintain the 0-3 range of value, adds the result to the base value for the table, and fetches the output code. The **AND**ed result is also stored back to **_PHASE.**

The outputs code table has values in both nibbles, i.e., 55,66,AA,99. The value left by **STEPCODE** is **AND**ed with either 000F or 00F0 to leave only the code for the selected motor. If 16-bit ports are available, this scheme can be readily expanded to control 4 motors.

I invented this method while working on an 1802-based application; since then I have seen it described three times in the literature, for control applications ranging from wafer scribers to radio telescopes, and it is completely in the public domain.

And last — regardless of the method used to generate the code, the **DO** loop which steps the motor should include a word which tests the limit switch in the direction of travel for the motor being stepped, and if set then executes **LEAVE**, and jumps immediately to **LOOP**, without stepping the motor.

*MTFBWY,*
*Wendall C. Gates, PE*
*Advanced Instrumentation*
*Santa Cruz, California*

*Thanks, Mr. Gates and MTFBWY, too.*
— Editor □

*"What did FORTH say to BASIC?"*
*"I'm okay, you're ready."*
—Ed Rotberg

# Start Your Own FIG Chapter

## What is a FIG Chapter?

There are two kinds of FIG chapters: local, and special-interest. Local chapters are centered in a city or region. special-interest chapters may be non-geographical; they focus on an interest area such as an application (e.g., robotics, telecommunication), or on FORTH for a particular computer.

All chapters must provide a contact point, and some form of regular public access (usually meetings). Non-geographical chapters will normally provide other forms of access, such as a newsletter or telecommunications, instead of meetings.

## Why Have a FIG Chapter?

A chapter lets you share information with other FORTH users in your geographical or application area. In addition, FIG provides several specific benefits:

(A) FIG will list your chapter in *FORTH Dimensions*, so that others can find your group.

(B) *FORTH Dimensions* will give priority to publishing chapter news, which can help you make professional contacts in the areas of your particular interests.

(C) FIG will occasionally supply material, such as meeting handouts or tapes, which can serve as a discussion topic at local meetings.

(D) FIG will supply its publications at bulk rates; local chapters can sell them to raise money, and to provide immedite local access to the material.

(E) Chapters can apply to FIG for one-time funding for activities.

## How to Start a FIG Chapter

To be recognized as a chapter, a group must have (1) a contact person, (2) regular public access (usually by meetings which are open to the public), and (3) at least five members of FIG. If you don't know five members in your area, FIG can help you contact them. If you want to start a chapter, send a request for a FIG Chapter Kit to the Chapter coordinator, FORTH Interest Group, P.O. Box 1105, San Carlos, CA 94070.

# Chapter News

### New Jersey Chapter

The New Jersey FIG Chapter held its fourth meeting October 28th at the Computer Center of Rutgers University, in New Brunswick. Meetings are purely informal discussions at this stage, with between ten and twenty members at our previous meetings; cocktail-party style multiple conversations is the rule, as we have yet to receive any offers for set piece presentations. Some presentations are in written handouts that may or may not be discussed at the meeting.

Everyone is a fig-FORTH model user at the moment, and several are involved in implementations on new machines including 8086, 68000, and Perken Elmer mainframes. As the number of participants grows a more formal organization may be created, but for the moment there is no mailing list or dues; attendees at one meeting set the time and place of the next meeting.

We have been very fortunate to date by having members volunteer facilities at their academic institutions for the meetings. Meetings are bimonthly. Call George Lyons at home at (201) 451-2905 to learn the place and time of the next meeting.

### Potomac Chapter

Joel Shprentz described and demonstrated the cross compiler system created by Nautilus Systems. Joel has used it to prepare new versions of FORTH for the TRS-80 and to create ROMable code for control applications.

The demonstration traced the development of an application from interactive development to cross compilation to ROM burning.

### Dayton Chapter

The Dayton Chapter of FIG held its second meeting at the Datalink Computer Center on Sept. 14, 1982 with fourteen members in attendance.

Mr. James Gaston told about his problems in learning FORTH when he was first programming. He suggested that a good club project would be to build a FORTH model for the new Motorola 68000 microprocessor. The membership was in favor of the project, so we will begin at our next meeting, October 12, 1982, with Jim offering suggestions in order to give each person a chance to learn the process.

# 79-FORTH ROM for Apple II

*Dr. C. H. Ting*

## The Design Goals

The main purpose of this project was to implement a FORTH system of the lowest possible cost, and to carry this exciting language to the large population of Apple II users. To lower the system cost, it is necessary to fully utilize all the existing resources inside the Apple II computer, without such expensive peripherals as the floppy disk drives. The design goals were thus set as follows:

- Use a stripped Apple II as the host
- Put the FORTH dictionary in 8K bytes of ROM
- Implement the 79-Standard with editor and assembler extensions
- Build a pseudo disk in RAM with cassette tape as the off-line storage medium

The result will be a FORTH computer in a small box, which can be operated standing-alone and has the capacity of expanding into many educational and professional applications.

## Development Tools

I did not have very sophisticated tools to develop 6502 based microcomputer systems. The only tool was a HP 65000 Development System, which had a 6502 cross assembler in it. The only way to build a FORTH system was to assemble the 6502 assembly source program on this development system, burn the object codes into a set of PROMs, and insert the PROMs into the Apple for testing and debugging. Any uncovered bug would have to be fixed at the source level. However, I had both the Auto Start Monitor and the Old Monitor in the Apple. The latter was very useful in the debugging process because of its trace capability.

## Approach

Because of the lack of good development tools, it would be very difficult trying to build a FORTH system from scratch towards the design goals. The best approach was to divide the project into two phases:

- Implement a fig-FORTH system using the 6502 fig-FORTH source listing; and
- Modify the fig-FORTH to meet the design goals.

It was extremely important to build a working fig-FORTH system, because the object codes can be checked out by comparing byte-by-byte with the source. This greatly eased the task of debugging. Once I had the fig-FORTH running, further modifications could be checked and debugged using the FORTH interpreter, which was much more convenient to use and test.

## Implementation

I first keyed in the 6502 source codes, identical to the 6502 fig-FORTH Source Listing. Both the source codes and the assembled object codes were thoroughly checked out. After changing the terminal I/O routines and offseting the object codes to start at 6000H, the resulting object codes were burnt into 2716's and moved into the Apple on the Apple ROM Card. Using the Apple Monitor, I could move the dictionary from the PROM's into RAM area, starting at 6000H. Debugging in RAM was much easier than doing it in ROM. The tracing aids provided by the fig-FORTH was helpful. However, I found it was more convenient to replace the JMP W-1 instruction by BRK, which returned the system to the monitor. To continue execution, I just keyed in OBOG in the monitor, which jumped over to W-1 (address OBOH) and continued onto the next word.

*The result was four 2716 PROMs sitting neatly on a small PC board.*

Only minor errors were detected and fixed at this stage, because most errors were flushed out by byte comparison of object codes. Since the entire system was in RAM, errors were corrected immediately and more tests could be carried out before a new set of PROMs were burnt.

After the fig-FORTH was thoroughly bug free, I proceeded to the task of modifications. The first thing to do was to trim the fig tree, making room for the editor and the assembler. I deleted the name fields and the link fields of all the run-time codes and some system words which the users are not expected to use. All the disk words were also deleted because the final system would not have a disk. The pseudo disk was implemented by a simple redefinition of **BLOCK**:

```
: BLOCK  ( n --- addr )
  MAXBUF MOD B/BUF * FIRST + ;
```

It returns a RAM address of the desired block, from which data can be referred.

The second task was to make the FORTH system ROMmable. All variables were either eliminated or changed to user variables. The only impure words were the vocabulary words like **FORTH**. To make **FORTH**, **EDITOR**, and **ASSEMBLER** stay comfortably in ROM, a new defining word **ROM-VOCABULARY** ought to be used:

*Continued on next page*

## 79-FORTH ROM for Apple II (continued from page 6)

```
: ROM-VOCABULARY ( addr --- )
  CREATE , DOES> @ CURRENT ! ;
HEX 1062 ROM-VOCABULARY FORTH
    1068 ROM-VOCABULARY EDITOR
    106E ROM-VOCABULARY ASSEMBLER
```

The addresses specified above point to the RAM locations where the name field addresses of the last words defined in the respective vocabularies are stored. These RAM locations are to be initialized at boot-up to:

```
FORTHLINK:  1060H:  A081
                    EF3D
EDITRLINK:  1066H:  A081
                    E8DF
ASSEMLINK:  106CH:  A081
                    EE01
```

The original **VOCABULARY** remained in the dictionary for the purpose of creating new vocabularies in RAM by the user.

The editor was basically the fig-FORTH editor. However, the command structure was modified to that used by Brodie in his *Starting FORTH.* I hoped to use *Starting FORTH* as an instruction manual for this ROM FORTH system and a compatible editor would not do any harm. The only major departure from the *Starting FORTH* editor was the handling of null strings. Only one string buffer (**PAD**) was used here, while Brodie used two independent buffers for searching and inserting.

Bill Ragsdale's 6502 Assembler was included in this FORTH to let the user experiment at the code level.

Finally, the whole system was updated to the 79-Standard. Many words needed to have their names changed. A few new words were added, and a few words needed to be redefined. Bob Smith's *'79-FORTH Conversion* was most helpful in this phase.

### Result

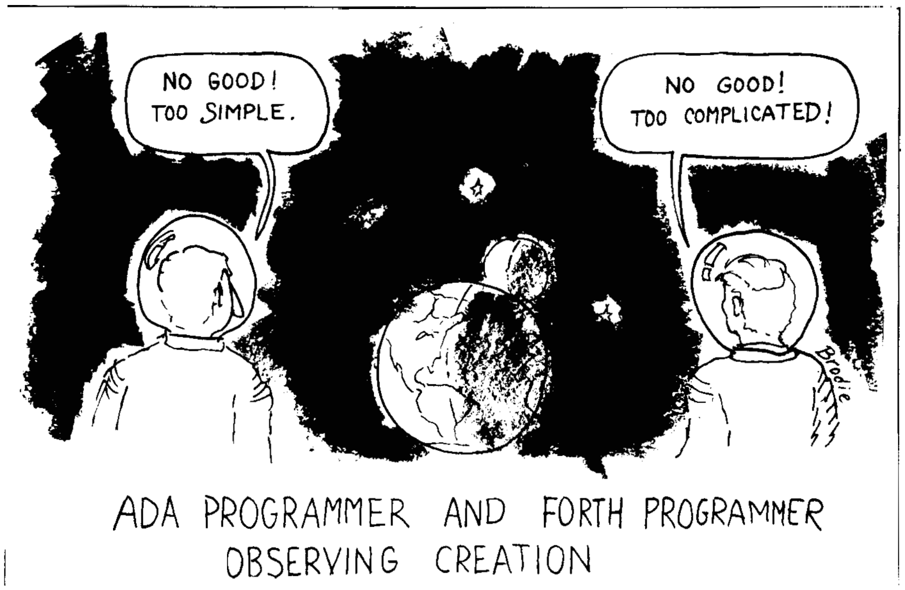
The result was four 2716 PROMs sitting neatly on a small PC board. When it was inserted into Slot 0 in an Apple II, it turned the Apple into a very powerful FORTH computer. In an Apple II with 48K bytes RAM, 24K are used as a pseudo disk which holds lots of programs. With some tricks like

```
-32 OFFSET !
```

one could even turn the 16K high graphics memory into a second disk, making the total disk memory 40K. The programs can be dumped to cassette tape for storage. By loading or dumping large chunks of memory from or to tape, the necessity of disk can be avoided while still having all the advantages of FORTH. The main dictionary, securely stored in PROMs, makes the system immune from frequent crashes during program development and testing. □

## FORTH Dementia

Leo Brodie


NO GOOD! TOO SIMPLE.
NO GOOD! TOO COMPLICATED!
Brodie
ADA PROGRAMMER AND FORTH PROGRAMMER OBSERVING CREATION

**FOR TRS-80 MODEL I OR III**
**IBM PERSONAL COMPUTER**

- ★ **MORE SPEED**
  10-20 times faster than interpreted BASIC.
- ● **MORE ROOM**
  Very compact compiled code plus VIRTUAL MEMORY makes your RAM act larger. Variable number of block buffers. 31-char.-unique wordnames use only 4 bytes in header!
- ★ **MORE INSTRUCTIONS**
  Add YOUR commands to its 79-STANDARD-plus instruction set!
  Far more complete than most Forths: single & double precision, arrays, string-handling, clock, graphics (IBM low-res. gives B/W and 16 color or 200 tint color display)
- ● **MORE EASE**
  Excellent full-screen Editor, structured & modular programming
  Word search utility
  THE NOTEPAD letter writer
  Optimized for your TRS-80 or IBM with keyboard repeats upper/lower case display driver, full ASCII
- ● **MORE POWER**
  Forth operating system
  Concurrent Interpreter AND Compiler
  VIRTUAL I/O for video and printer, disk and tape (10-Megabyte hard disk available)
  Full 8080 or 8088 Assembler aboard
  (Z80 Assembler also available for TRS-80)
  Intermix 35- to 80-track disk drives
  IBM can read, write and run M.3 Disks
  M.3 can read, write and run M.1 disks

**MMSFORTH**

**THE PROFESSIONAL FORTH SYSTEM FOR TRS-80 & IBM PC**

(Thousands of systems in use)

MMSFORTH Disk System (requires 1 disk drive, 32K RAM)
V2.0 For Radio Shack TRS-80 Model I or III .......... **$129.95**
V2.1 For IBM Personal Computer (80-col. screen) .... **$249.95**

**AND MMS GIVES IT PROFESSIONAL SUPPORT**

Source code provided
MMSFORTH Newsletter
Many demo programs aboard
MMSFORTH User Groups
Inexpensive upgrades to latest version
Programming staff can provide advice, modifications and custom programs, to fit YOUR needs.

MMSFORTH UTILITIES DISKETTE: includes FLOATING POINT MATH (BASIC ROM routines plus Complex numbers, Rectangular-Polar coordinate conversions, Degrees mode, more); a powerful CROSS-REFERENCER to list Forth words by block and line; plus (TRS-80) a full Forth-style Z80 assembler requires MMSFORTH V2.0, 1 drive & 32K RAM) ....... **$39.95***

FORTHCOM: communications package provides RS-232 driver, dumb terminal mode, transfer of FORTH blocks, and host mode to operate a remote FORTHCOM systems (requires MMSFORTH V2.0, 1 drive & 32K RAM) .............. **$39.95***

THE DATAHANDLER: a very fast database management system operable by non-programmers (requires MMSFORTH V2.0, 1 drive & 32K RAM) ......................... **$59.95***

FORTHWRITE: fast, powerful Word Processor w/easy keystrokes, Help screens, manual & demo files. Full proportional w/tabs, outdenting. Include other blocks, documents & keyboard inputs— ideal for form letters (requires MMSFORTH V2.0, 2 drives & 48K RAM) ...................... **$175.00***

MMSFORTH GAMES DISKETTE: real-time graphics & board games w/source code. Includes BREAKFORTH, CRASHFORTH, CRYPTOQUOTE, FREEWAY (TRS-80), OTHELLO & TICTACFORTH (requires MMSFORTH V2.0, 1 drive & 32K RAM) **$39.95***

**Other MMSFORTH products under development**

**FORTH BOOKS AVAILABLE**

MMSFORTH USERS MANUAL - w/o Appendices ..... **$17.50***

STARTING FORTH - best! .......................... **$15.95***

THREADED INTERPRETIVE LANGUAGES - advanced, analysis of FORTH internals ............................ **$18.95***

PROGRAM DESIGN & CONSTRUCTION - intro. to structured programming, good for Forth ...................... **$16.00***

FORTH-79 STANDARD MANUAL - official reference to 79-STANDARD word set, etc ........................ **$13.95***

FORTH SPECIAL ISSUE, BYTE Magazine (Aug. 1980) - A collector's item for Forth users and beginners ............. **$4.00***

* ORDERING INFORMATION: Software prices include manuals and require signing of a single computer license for one-person support. Describe your Hardware. Add $2.00 S/H plus $3.00 per MMSFORTH and $1.00 per additional book; Mass. orders add 5% tax. Foreign orders add 20%. UPS COD, VISA and M/C accepted; no unpaid purchase orders or refunds.

*Send SASE for free MMSFORTH information*
*Good dealers sought*

Get MMSFORTH products from your computer dealer or

**MILLER MICROCOMPUTER SERVICES (B9)**
61 Lake Shore Road, Natick, MA 01760
(617) 653-6136

# THE FORTH SOURCE™

**MVP-FORTH** - A Public Domain Product

MVP Forth is fig-FORTH modified by 100% of the FORTH-79 Standard Required Word Set plus the vocabulary for the instructional book *Starting FORTH*. Editor, assembler and utilities are included.

Transportability of programs is assured since the kernal of MVP-FORTH is the same for all computers to the machine dependent READ/WRITE instructions.

Modification and extension (up or down) is simplified by having the source code and through the use of MVP-FORTH Programming Aids and Cross Compilers.

The CP/M® are supplied on 8", SS/SD, IBM 3740, format disks. The include a track and sector calculation array for down loading to other sizes and formats. Other disks are machine specific.

*All About FORTH* is an annotated glossary of MVP-FORTH words as well as other dialects. It is in 8080 code, other MVP-FORTH implementations include documentation of the differences between it and other CPU's and computers.

**MVP-FORTH PRODUCTS** for CP/M® IBM-PC® and Apple®

- ☐ **MVP-FORTH Programmer's Kit** including disk with documentation, ALL ABOUT FORTH, and STARTING FORTH. Assembly source listing versions. $100
- ☐ **MVP-FORTH Disk** with documentation. Assembly source listing version. $75
- ☐ **MVP-FORTH Cross Compiler** with MVP-FORTH source in FORTH. $300
- ☐ **MVP-FORTH Programming Aids** for decompiling, callfinding, and translating. Specify computer. $150
- ☐ **MVP-FORTH Fast Floating Point** for Apple II/II+ on board *with 9511 math chip. Requires MVP-FORTH for Apple* $400
- ☐ **MVP-FORTH Assembly Source** Printed listing. $20
- ☐ **ALL ABOUT FORTH** by Haydon. MVP-FORTH reference, plus fig-FORTH and FORTH-79. $20

★★★MVP-FORTH operates under a variety of CPU's, computers, and operating systems. Specify your computer and operating system. CP/M supplied on 8", SS/SD, 3740 format.★★★

## FORTH DISKS

**FORTH** with editor, assembler, and manual.

- ☐ **APPLE II/II+** by MicroMotion $100
- ☐ **APPLE II** by Kuntze $90
- ☐ **ATARI®** valFORTH $50
- ☐ **CP/M®** by MicroMotion $100
- ☐ **CROMEMCO®** by Inner Access $100
- ☐ **HP-85** by Lange $90
- ☐ **IBM-PC®** by Laboratory Microsystems $100
- ☐ **NOVA** by CCI, quad floppy $100
- ☐ **PET®** by FSS $90
- ☐ **TRS-80/I®** by Nautilus Systems $90
- ☐ **6800** by Talbot Microsystems $100
- ☐ **6809** by Talbot Microsystems $100
- ☐ **Z80** by Laboratory Microsystems $50
- ☐ **8086/88** by Laboratory Microsystems $100
- ☐ **VIC FORTH** byHES, VIC20 cartridge $60

**Enhanced FORTH** with: F-Floating Point, G-Graphics, T-Tutorial, S-Stand Alone, M-Math Chip Support, MT-Multi-Tasking, X-Other Extras, 79-FORTH-79.

- ☐ **APPLE II/II+** by Micro-Motion, F, G, & 79 $140
- ☐ **ATARI** by PNS, F, G, & X. $90
- ☐ **CP/M** by MicroMotion, F & 79 $140
- ☐ **Apple II/II+, GraFORTH** by Insoft, stand alone graphics $75
- ☐ **H89/Z89** by Haydon, T & S $250
- ☐ **H89/Z89** by Haydon, T $175
- ☐ **IBM-PC, PolyFORTH** by FORTH Inc., F, G, S, M, MT, & X $300
- ☐ **Multi-Tasking FORTH** by Shaw Labs, CP/M, X & 79 $395
- ☐ **TRS-80/I or III** by Miller Microcomputer Services, F, X, & 79 $130
- ☐ **TUTORIAL** by Laxen & Harris, CP/M with a copy of *Starting FORTH* $95
- ☐ **Extensions** for Laboratory Microsystems IBM,Z80, and 8086
  - ☐ Software Floating Point $100
  - ☐ *8087 Support* (IBM-PC or 8086) $100
  - ☐ 9511 Support (Z80 or 8086) $100
  - ☐ Color Graphics (IBM-PC) $100
  - ☐ Data Base Management $200

**CROSS COMPILERS** Allow extending, modifying and compiling for speed and memory savings, can also produce ROMable code. •Requires FORTH disk.

| | | | |
|---|---|---|---|
| ☐ CP/M | $200 | ☐ IBM• | $300 |
| ☐ H89/Z89 | $200 | ☐ 8086• | $300 |
| ☐ TRS-80/I | $200 | ☐ Z80• | $300 |
| ☐ Northstar® | $200 | ☐ Apple II/II+ | $350 |

☐ **fig-FORTH Programming Aids** for decompiling, callfinding, and translating. Specify CP/M, IBM-PC, 8086, Z80, or Apple II/II+ $150

## FORTH MANUALS, GUIDES & DOCUMENTS

- ☐ **ALL ABOUT FORTH** by Haydon. An annotated glossary of common FORTH words. MVP-FORTH reference. $20
- ☐ ***And So FORTH*** by Huang. A college level text. $25
- ☐ **FORTH Encyclopedia** by Derick & Baker. A complete programmer's manual to fig-FORTH with FORTH-79 references. Flow charted, 2nd Ed. $25
- ☐ **Starting FORTH** by Brodie. Best instructional manual available. (soft cover) $16
- ☐ **Starting FORTH** (hard cover) $20
- ☐ **1980 FORML Proc.** $25
- ☐ **1981 FORML Proc.** 2 Vol. $40
- ☐ **1982 FORML Proc.** $25
- ☐ **1981 Rochester FORTH Proc.** $25
- ☐ **1982 Rochester FORTH Proc.** $15
- ☐ **Using FORTH** $25
- ☐ **A FORTH Primer** $25
- ☐ **Threaded Interpretive Languages** $20
- ☐ **AIM FORTH User's *Manual*** $12
- ☐ **APPLE User's Manual** MicroMotion $20
- ☐ **TRS-80 User's Manual,** MMSFORTH $19
- ☐ **METAFORTH** by Cassady. Meta compiler in 8080 code $30
- ☐ ***Systems Guide to fig-FORTH*** $25
- ☐ **Caltech FORTH Manual** $12
- ☐ **Invitation to FORTH** $20
- ☐ **PDP-11 FORTH User's Manual** $20
- ☐ **CP/M User's Manual,** MicroMotion $20
- ☐ **FORTH-79 Standard** $15
- ☐ **FORTH-79 Standard Conversion** $10
- ☐ **Tiny Pascal in fig-FORTH** $10
- ☐ **NOVA fig-FORTH** by CCI with editor, assembler, and utilities $15
- ☐ **MVP-FORTH** Source Listings ☐ IBM-PC ☐ CP/M ☐ Apple II/II+ $20

☐ **Installation Manual for fig-FORTH,** contains FORTH model, glossary, memory map and instructions $15

**Source Listings of fig-FORTH,** for specific CPU's and computers. The Installation Manual is required for implementation. Each $15

| | | | |
|---|---|---|---|
| ☐ 1802 | ☐ 6502 | ☐ 6800 | ☐ AlphaMicro |
| ☐ 8080 | ☐ 8086/88 | ☐ 9900 | ☐ APPLE II |
| ☐ PACE | ☐ 6809 | ☐ NOVA | ☐ PDP-11/LSI-11 |
| ☐ 68000 | ☐ Eclipse | ☐ VAX | |

**Ordering Information:** Check, Money Order (payable to MOUNTAIN VIEW PRESS, INC.), VISA, MasterCard or COD's accepted. No billing or unpaid PO's. California residents add sales tax. Shipping costs in US included in price. Foreign orders, pay in US funds on US bank, include for handling and shipping by Air: $5 for each item under $25, $10 for each item between $25 and $99 and $20 for each item over $100. Minimum order $10. All prices and products subject to change or withdrawal without notice. Single system and/or single user license agreement required on some products.

**DEALER & AUTHOR INQUIRIES INVITED**

**MOUNTAIN VIEW PRESS, INC.**

**PO BOX 4656 MOUNTAIN VIEW, CA 94040 (415) 961-4103**

# ROMable FORTH with Separate Headers

*Robert H. Hertel*
*Robert D. Villwock*
*Microsystems, Inc.*

Traditional implementations of FORTH assume a single computer environment in which the program is compiled, tested and used. In this environment it is expected that:

- Memory is all-RAM
- Terminal and disk are available
- System software (compiler, editor, etc.) is needed

In a typical dedicated microprocessor application such as a "smart" product of some kind, there are two distinct environments which we will call the *development* and *target* environments. The development environment is usually the same as the traditional FORTH environment, but the target environment is different:

- ROM/RAM memory organization
- No conventional terminal or disk
- System software is not needed.

In our own work, the development setup consists of a microprocessor development system connected to a prototype target system by means of an in-circuit emulator. The memory space can be mapped so that some parts of the memory reside in the development system and other parts in the target hardware.

Until recently, the usual method of developing FORTH software for dedicated applications has been to compile and test with a traditional FORTH implementation. After testing is completed in this mode, a "target compiler" or cross compiler is used to reprocess the source code to produce the final target code. This produces headerless target code without the name fields and without unneeded system software. ROM copies of this code can be installed in the target system.

The cross-compilation method of producing ROMable code has several drawbacks. Obviously it adds an extra step to the software development process. The cross compiler itself is complex and much more difficult to use than an ordinary compiler. With the target names discarded, interactive testing becomes difficult or impossible at a critical point in the development process.

When we selected FORTH as our primary software development tool, we decided to develop a new compiler to recognize and fully support the dual development/target environments. In the resulting implementation (called proFORTH™) the ROM/RAM and system/target separations are made as the code is compiled. Target names are retained in the system area of memory in such a way that interactive testing can continue until and even after the target code is transferred to ROM. When development is completed it is only necessary to program ROMs for the appropriate memory areas; system code is simply discarded.

The remainder of this discussion will describe those elements of the proFORTH™ system related to compiling ROMable target code, and will explain how the programmer uses the system.

## Memory Organization

The processor's memory space is divided into three kinds of *segments:*

- ROM — target code or read-only memory
- RAM — target variables or read/write memory
- DICTIONARY — system code, names, etc. (also read/write)

Any number of segments of each kind may be defined. ROM and RAM segments are dictated by the needs of the application and the design of its hardware. For example, separate ROM segments might be assigned for an interrupt vector jump table, optional code, and for all other target code. RAM areas might be set up for a CRT refresh memory and for all other target RAM. Figure 1 shows a memory map with these ROM and RAM segments defined.

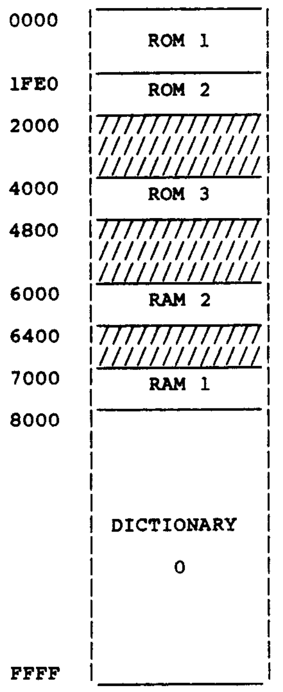


FIGURE 1 - SAMPLE MEMORY MAP

Often a single dictionary area will suffice, but proFORTH™ supports multiple dictionaries and they can be used in a variety of ways. For now we will assume that only one dictionary is needed.

Segments within each class are numbered as indicated in Figure 1. ROM and RAM numbers begin with 1; dictionaries are numbered from zero. They are established using the words **ROM, RAM** and **DICTIONARY**. Thus the directives

**8000 FFFF 0 DICTIONARY**
**6000 63FF 2 RAM**
**4000 47FF 3 ROM**

assign the address space 8000 through FFFF to dictionary 0, 6000 through 63FF to RAM segment 2 and 4000 to 47FF to ROM segment 3 (we will assume throughout this article that numbers are expressed in hexadecimal). More specifically, each memory segment has a *segment control block* structure like the one for

*Continued on next page*

RAM 2 shown in Figure 2. The control block has three pointers referred to as **VBOT**, **VLOC** and **VTOP** (V for variable memory). **VBOT** and **VTOP** point to the lower and upper address limits of the RAM segment, and **VLOC** points to the next available address within the segment. In the example above, execution of the sequence **6000 63FF 2 RAM** sets the contents of **VBOT** and **VLOC** in RAM control block 2 to 6000, and the contents of **VTOP** to 63FF. As space in RAM segment 2 is used, the contents of **VLOC** will be incremented so that it always points to the next available byte.

The dictionary and ROM control blocks have the same form as the RAM control blocks, except that the pointers are *referred to as* **DBOT**, etc., and **CBOT**, etc. (C stands for *code*).

## Definition Components

Although for some processors the standard proFORTH™ implementation uses direct threaded code for faster execution and byte savings, this

*Continued on next page*

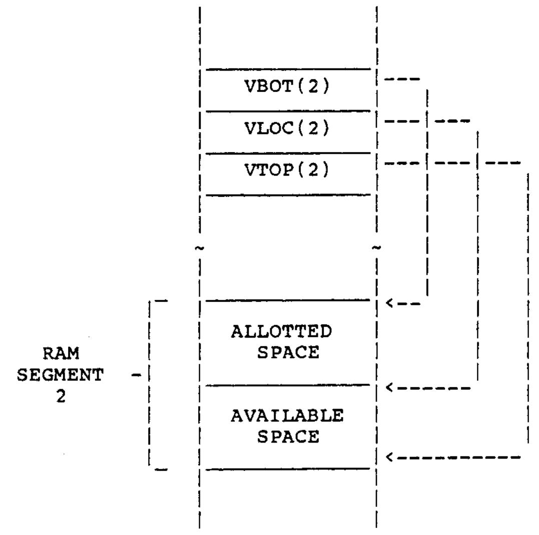


FIGURE 2 - SEGMENT CONTROL BLOCK
FOR RAM SEGMENT 2

-=-=-=-=-

-=-=-=- 8080/Z80 FIG-FORTH for CP/M & CDOS systems -=-=-=-

-=-=-=-=-

$50 saves you keying the FIG FORTH model and many published FIG FORTH screens onto diskette and debugging them. You receive TWO 8 inch diskettes (single sided, single density, soft sectored only). The first disk is readable by Digital Research CP/M or Cromemco CDOS and contains 8080 source I keyed from the published listings of the FORTH INTEREST GROUP (FIG) plus a translated, enhanced version in ZILOG Z80 mnemonics. This disk also contains executable FORTH.COM files for Z80 & 8080 processors.

-=-=-=-=-

The 2nd disk contains FORTH readable screens including a extensive FULL-SCREEN EDITOR plus many items published in FORTH DIMENSIONS, including a FORTH TRACE utility, a model data base handler, an 8080 ASSEMBLER and formatted memory dump and I/O port dump words. The disks are packaged in a ring binder along with a complete listing of the FULL-SCREEN EDITOR and the FIG-FORTH INSTALLATION MANUAL (the language model of FIG-FORTH, a complete glossary, memory map, installation instructions and the FIG line editor listing and instructions).

-=-=-=-=-

This entire work is placed in the public domain in the manner and spirit of the work upon which it is based. Copies may be distributed when proper notices are included.

-=-=-=-=-

```
                                                   USA   Foreign
                                                           AIR
+-+
| |   Above described package ...................$50     $60
+-+
+-+
| |   Printed Z80 Assembly listing w/ xref........$15     $18
+-+          (Zilog mnemonics)
+-+
| |   Printed 8080 Assembly listing..............$15     $18
+-+
                                       TOTAL  $_____    _____
```

-=-=-=-=-

Price includes postage. No purchase orders without check. Arizona residents add sales tax. Make check or money order in US Funds on US bank, payable to:

Dennis Wilson c/o
Aristotelian Logicians
2631 East Pinchot Avenue
Phoenix, AZ 85016
(602) 956-7678

-=-=-=-=-

discussion will consider only the more familiar indirect threaded code. Figure 3(a) shows the proFORTH™ definition structure for a system constant — that is, a constant which is not referenced in high-level target code. Except for the location of the LFA, it is essentially identical to the traditional structure, in that the entire definition is compiled into the active dictionary (in this case, dictionary 0). The header begins at the link field address or LFA, and consists of the link and the definition's name. The *body* immediately follows the header in memory and consists in this case of a pointer to a common support procedure named **(CONST)** located at the CFA or code field address, followed by the parameter field at the parameter field address of PFA. In the example the parameter field contains the constant's value, which the procedure **(CONST)** will put on the stack when it executes. In proFORTH™ the procedure **(CONST)** resides in target ROM so that it is also available for supporting target constants.

Consider now the structure for a *target* constant shown in Figure 3(b). Notice that the definition body is now located in target ROM. In order to be able to access the body, a pointer is added to the header at the definition field address DFA, and the CFA of the definition is stored there.

*Continued on next page*

```
                |‾       LFA   |‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|        ‾|
                |              |  LINK          |         |
                |              |‾‾|‾‾‾‾‾‾‾‾‾‾‾‾‾|         |
    HEADER     -|        NFA   |  |  LENGTH     |         |
                |              |‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|         |
                |              |                |         |-   ACTIVE
                |              |  NAME          |         |   DICTIONARY
                |_             |                |         |
                |‾   DFA/CFA   |‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|         |
    BODY       -|              |‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|---      |
                |        PFA   |  VALUE         |   |     |
                |_             |________________|   |    _|
                                                    |
                                                    |
                |‾   (CONST)   |‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|<--'    ‾|
   SUPPORT     _|              |                |         |-   ACTIVE
   PROCEDURE    |              |                |         |  ROM SEGMENT
                |_             |________________|        _|
```

FIGURE 3(a) - proFORTH DICTIONARY STRUCTURE FOR SYSTEM CONSTANT

```
                |‾       LFA   |‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|        ‾|
                |              |  LINK          |         |
                |              |‾‾|‾‾‾‾‾‾‾‾‾‾‾‾‾|         |
                |        NFA   |  |  LENGTH     |         |
                |              |‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|         |
    HEADER     -|              |                |         |-   ACTIVE
                |              |  NAME          |         |   DICTIONARY
                |              |‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|         |
                |        DFA   |                |         |
                |_             |________________|---     _|
                                                    |
                                                    |
                |‾       CFA   |‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|<--'    ‾|
    BODY       -|              |‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|---      |
                |        PFA   |  VALUE         |   |     |
                |_             |________________|   |     |
                                                    |     |_   ACTIVE
                                                    |     |  ROM SEGMENT
                |‾   (CONST)   |‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾|<--'     |
   SUPPORT     _|              |                |         |
   PROCEDURE    |              |                |         |
                |_             |________________|        _|
```

FIGURE 3(b) - proFORTH DICTIONARY STRUCTURE FOR TARGET CONSTANT

**Look to TIMIN Engineering**

for FORTH software of professional quality.

*ready to-run **FORTH development systems**

***application programs** in FORTH

***Consulting services**, including custom program development

Our latest product:
**DUAL TASKING FORTH**

Now you can run your process control programs in background while still using your FORTH system in the normal way. Background and foreground tasks may each be written in high level FORTH. They can execute simultaneously and exchange data. The foreground task can control the background task.

**Available NOW:**
8" diskette $285

**Write for our FORTH information booklet**

**Timin Engineering Co.**
6044 Erlanger St.
San Diego,
CA 92122
(714) 455-9008

Another pair of examples is given in Figures 4(a) and 4(b) to show the structure of definitions for system and target variables. Again, the system variable's definition is similar in form to the traditional one. The CFA points to the generic run-time procedure for variables called **(VAR)**. Since the two-byte space for the variable is allocated in line, the procedure **(VAR)** is usually used only for system-based (dictionary) variables and is itself therefore, stored in system memory.

Figure 4(b) shows how proFORTH™ separates the definition components for a *target* variable. The system component is the header — essentially, the name. A DFA pointer is added to it, and in it is stored the CFA or body address. The body is compiled in target ROM. However, instead of the generic procedure **(VAR)**, the CFA of the ROM-based body is pointed to **(CONST)**. In addition, the space allocated for the variable will be in the target RAM area. The address of the RAM space allotted is stored at the PFA in ROM, i.e., it is the *value* of the constant. When the variable's name is typed, the **(CONST)** support procedure executes, placing the variable's address on the stack in the usual way.

Other definition structures follow the same principles as those discussed. For example, a code word is similar in structure to the constant, except that the CFA points to the PFA, and the assembly language code begins at the PFA. A colon definition's CFA points to the colon support procedure **(COLON)**, and at the PFA is compiled the list of addresses of pointers to executable code which will be processed by the address interpreter at run time. All these structures can be compiled "in line" or in target ROM (connected by a DFA pointer).

Extensions involve more complex structures than those illustrated, but to avoid obscuring the principles in a mass of details, we will concentrate on the simpler definition forms.

## Compilation Control

The first step in compiling proFORTH™ code is to define the ROM,

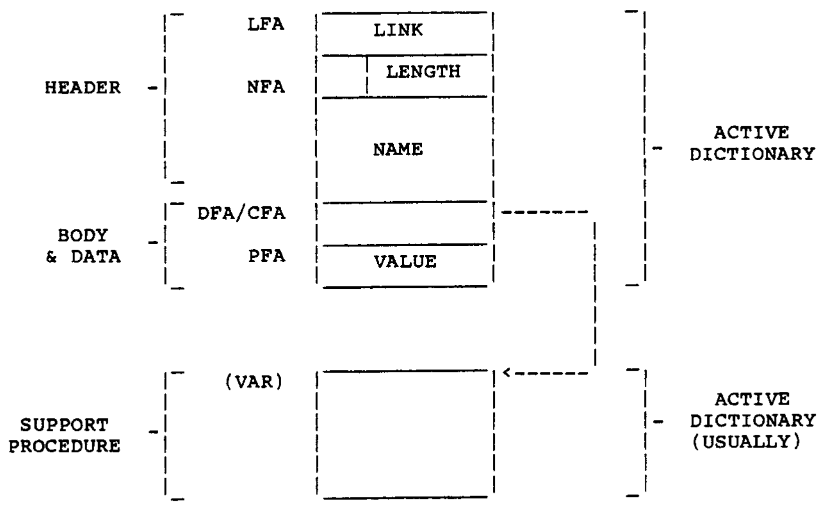


FIGURE 4(a) - proFORTH DICTIONARY STRUCTURE FOR SYSTEM VARIABLE

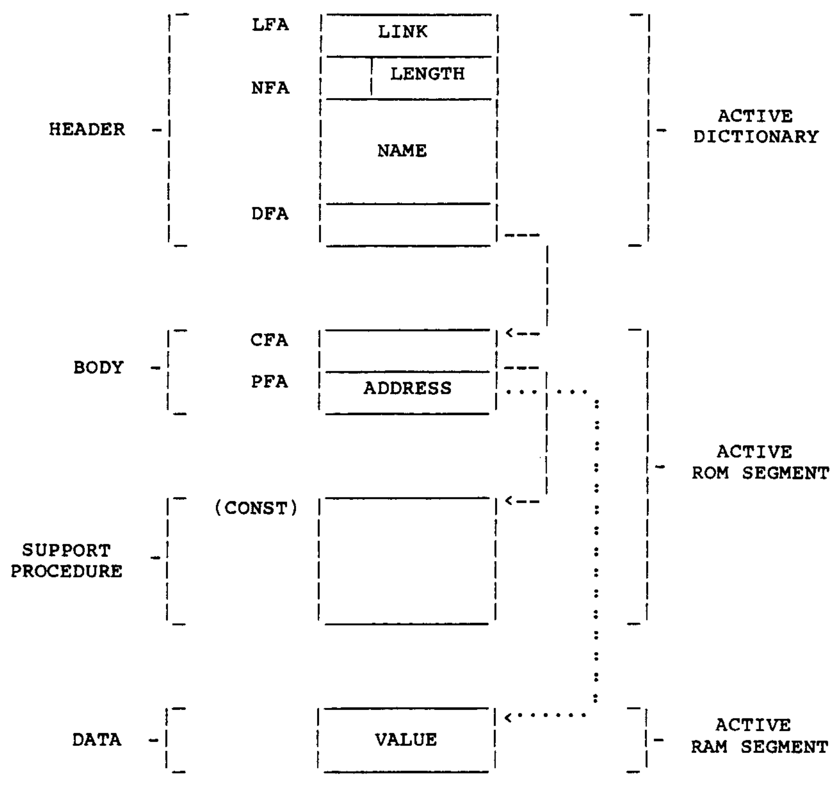


FIGURE 4(b) - proFORTH DICTIONARY STRUCTURE FOR TARGET VARIABLE

RAM and dictionary segments as explained above. We will assume that the basic proFORTH™ system has already been supplied or compiled. This sytem will have two main components: the system code, and the target code nucleus. The system code (located in dictionary 0 in our example) includes the text interpreter, compiler, disk interface, assembler and editor, as well as the names of all accessible system and target words. The target nucleus (which would be located in ROM segment 1 in our illustration) includes the bodies of all words which will be needed by the target system, as well as the address interpreter and support procedures like **(CONST)** and **(COLON)**. Stacks and variables needed by the target are allocated in target RAM — say RAM segment 1.

```
0 DSEG  0 CSEG               ( compile headers & bodies in dict. 0 )

8 CONSTANT SYS.CON           ( define a system constant equal to 8 )

0 VSEG                       ( allocate variable space in dict. 0 )

VARIABLE SYS.VAR             ( define a system variable )

: H.                         ( define a system procedure )
  BASE @  HEX SWAP .
  BASE ! ;

1 CSEG                       ( compile bodies in ROM seg. 1 )

1 VSEG                       ( allocate variable space in RAM seg. 1 )

9 CONSTANT TARG.CON          ( define a target constant equal to 9 )

VARIABLE TARG.VAR            ( define a target variable )

: 2*    DUP + ;              ( define a target procedure )
```

FIGURE 5 - EXAMPLES OF DIRECTIVES FOR COMPILATION CONTROL

During compilation, there will at all times be an active dictionary, ROM segment and RAM segment. These are specified by the compiler directives **DSEG**, **CSEG** and **VSEG** respectively. For instance, the sequence

**0 DSEG 1 CSEG 1 VSEG**

activates ROM and RAM segments 1 and dictionary 0. With these assignments, the compiler will separate definition components as in Figures 3(b) and 4(b):

**DICTIONARY 0–headers**
**ROM segment 1–bodies**
**RAM segment 1–variable space allocations**

Recall that there are no target memory segments corresponding to the zero ROM and RAM indices. Instead, these refer to the active *dictionary*, and are used for compiling system code. Thus **0 CSEG** instructs the compiler to compile definition bodies into the active dictionary. In the same way **0 VSEG** causes the compiler to make RAM allocations in the active dictionary. The sequence

**0 DSEG 0 CSEG 0 VSEG**

then causes *all* definition components to be compiled into dictionary 0, and proFORTH™ functions exactly like an ordinary FORTH compiler. This is in fact the default mode of proFORTH™. When **CSEG 0** is active, the DFA pointer is not needed (since the CFA immediately follows the header) and is omitted as shown in Figures 3(a) and 4(a). One of the control bits in the length byte is used to indicate when the DFA pointer is omitted and the code is in line.

As proFORTH™ code is compiled, there are effectively three compiler pointers replacing the usual dictionary pointer **DP**. From the earlier discussion, we see that these pointers are just the **DLOC**, **CLOC** and **VLOC** for the active segments. As space in a segment is used, the appropriate pointer is advanced. At each step it is checked against the segment's limits (e.g., **CBOT** and **CTOP**) and the operator is warned if the pointer exceeds its assigned boundaries.

Figure 5 shows the compiler directives used to compile such typical definitions as system and target constants, system and target variables, and system and target colon definitions. As the examples illustrate, there is no change at all in the source code for the definitions themselves; all that is necessary is to insert the appropriate **DSEG**, **CSEG** and **VSEG** directives.

## Absolute Compilation

There are some situations in which it is necessary to specify the exact location of a target code construct. For example, an interrupt system might use a jump table with the first jump located at **JBASE**. To support absolute compilation proFORTH™ uses one extra ROM segment control block. The absolute ROM segment limits are set to 0000 and FFFF. The words **ACSEG** (for absolute code segment) and **CSEG** are used to manipulate the absolute segment pointer (or **CLOC**). Their glossary definitions are:

| | |
|---|---|
| **ACSEG** ( addr --- b ) | Begin compiling in the absolute CSEG at address addr. Save index b of currently active CSEG on stack. |
| **CSEG** ( b --- ) | Select ROM segment b to be the recipient of future definitions and constant data. |

Returning now to our interrupt jump table example, consider the definitions

*Continued on next page*

in Figure 6. After assembling the code for the interrupt service routines **INT.0** through **INT.7**, an **ACSEG** directive is used to set the absolute compile pointer to **JBASE**. The active ROM segment index is saved on the stack. Next the actual jump table is assembled, starting at the address of **JBASE**. Finally, the **CSEG** directive restores the previously active ROM segment.

**HERE, ALLOT, etc.**

In traditional monolithic FORTH implementations, the word **HERE** provides the next available dictionary address (usually the contents of a variable called **DP**). In proFORTH™, the word **HERE** provides the next available address in the active ROM segment. For CSEG = 0, this defaults to the active dictionary segment. Similarly, the word **ALLOT** allocates space in the active ROM segment (or dictionary if CSEG = 0). In addition to **HERE**, proFORTH™ provides the word **VHERE** to obtain the next available address in the active RAM segment. If VSEG = 0, **VHERE** provides the next available address in the active dictionary just as for **HERE**. Therefore, in its default mode proFORTH™ remains completely compatible with traditional FORTH, but broadens in a consistently logical way as segment control directives are issued. This enables the user to have complete control over where the compiler stores each component of every definition.

**Conclusion**

It is not possible, in an article of this size, to even introduce all of the compilation control mechanisms available in proFORTH™. For example, we have not discussed the subjects of multiple dictionaries, extensions, selective purging of names, split system definitions, etc. Insofar as ROMable code compilation is concerned, the principles discussed above apply to all of these areas even though implementation details are often complex.

proFORTH™ logically extends and broadens the meaning of the traditional FORTH words such as **HERE** and **ALLOT**, so that the programmer is given complete control over compilation memory assignments. This is accomplished without giving up the traditional behavior of a monolithic dictionary system if the user wishes to program in that environment. □

```
CODE INT.0  ...  RET, END-CODE

CODE INT.1  ...  RET, END-CODE

    .

    .

    .

CODE INT.7  ...  RET, END-CODE

JBASE ACSEG
   ' INT.0 JMP,  ' INT.1 JMP,   ...  ' INT.7 JMP,
                                           CSEG
```

FIGURE 6 - EXAMPLE OF ABSOLUTE COMPILATION

# CHAIRMAN OF THE BOARDS

## INTRODUCING P-FORTH

The P-FORTH Card is the key member in a family of control systems cards offered by the innovators at Peopleware Systems, Inc. P-FORTH has four major advantages:

1. It is a versatile building block. The simple addition of a power supply and terminal makes the P-FORTH card both a low cost development system as well as a target system.
2. An integrated high-level interactive language allows for fast software development.
3. Application programs are stored automatically in non-volatile memory.
4. The STD BUS interface allows the use of a variety of existing peripheral cards.

## PUTTING P-FORTH TO WORK:

Users interactively develop applications through a combination of hardware and software. These applications are automatically programmed into non-volatile memory (EEROM). When the applications are proven and functioning, a single switch transforms the system from the developmental mode into the target system.

**PEOPLEWARE SYSTEMS INC.**

5190 West 76th St.

Mpls., MN 55435 USA

(612) 831-0827 • TWX 910-576-1735

## SOFTWARE

- An interactive high-level language following the fig-FORTH model
- A monitor for system checkout
- "FORTH-type" screen editor for developing application programs
- A "FORTH-type" assembler for writing assembly language routines
- High-level interrupt-linkage
- High-level communications protocol for down loading from a host system.

## HARDWARE

- 6801 microprocessor
- 6K EEROM
- 8K FORTH firmware
- 2K RAM
- STD BUS interface
- RS232 serial I/O
- 16 TTL I/O lines
- Programmable timer

# Z-80® and 8086 FORTH

## PC/FORTH™ for IBM® Personal Computer available now!

**FORTH Application Development Systems** include interpreter/compiler with virtual memory management, assembler, full screen editor, decompiler, demonstration programs, utilities, and 130 page manual. Standard random access disk files used for screen storage. Extensions provided for access to all operating system functions.

| | |
|---|---|
| **Z-80 FORTH** for CP/M® 2.2 or MP/M | $ 50.00 |
| **8086 FORTH** for CP/M-86 | $100.00 |
| **PC/FORTH** for IBM Personal Computer | $100.00 |

**Extension Packages** for FORTH systems

| | |
|---|---|
| Software floating point | $100.00 |
| Intel 8087 support (PC/FORTH, 8086 FORTH only) | $100.00 |
| AMD 9511 support (Z-80, 8086 FORTH only) | $100.00 |
| Color graphics (PC/FORTH only) | $100.00 |
| Data base management | $200.00 |

**Nautilus Cross-Compiler** allows you to expand or modify the FORTH nucleus, recompile on a host computer for a different target computer, generate headerless code, and generate ROMable code with initialized variables. Supports forward referencing to any word or label. Produces load map, list of unresolved symbols, and executable image in RAM or disk file. No license fee for applications created with the Cross-Compiler! Prerequisite: one of the application development systems above for your host computer.

Hosts: Z-80 (CP/M 2.2 or MP/M), 8086/88 (CP/M-86), IBM PC (PC/DOS or CP/M-86)
Targets: Z-80, 8080, 8086/88, IBM PC, 6502, LSI-11

| | |
|---|---|
| Cross-Compiler for one host and one target | $300.00 |
| Each additional target | $100.00 |

**FORTH Programming Aids** by Curry Associates. Includes Translator, Callfinder, Decompiler, and Subroutine Decompiler. 40 page manual. Used with Cross-Compiler to generate minimum size target applications.

| | |
|---|---|
| Specify host system | $150.00 |

**Z-80 Machine Tests** Memory, disk, console, and printer tests with all source code in standard Zilog mnemonics .......... $ 50.00

All software distributed on eight inch single density soft sectored diskettes, except PC/FORTH on 5¼ inch soft sectored single sided double density diskettes. Micropolis and North Star disk formats available at $10.00 additional charge.

Prices include shipping by UPS or first class mail within USA and Canada. Overseas orders add US$10.00 per package for air mail. California residents add appropriate sales tax. Purchase orders accepted at our discretion. No credit card orders.

**Laboratory Microsystems**
**4147 Beethoven Street**
**Los Angeles, CA 90066**
**(213) 306-7412**

Z-80 is a registered trademark of Zilog, Inc.
CP/M is a registered trademark of Digital Research, Inc.
IBM is a registered trademark of International Business Machines Corp.

# 8080 fig-FORTH in ROM

*Ted Croal*
*Brantford General Hospital*
*Ontario, Canada*

*Editor's note: This article outlines the author's experiences in modifying the 8080 fig-FORTH assembly listing to run in ROM. The easiest way to make such a substantive change to a FORTH system, of course, is to use a cross-compiler. This allows you to modify FORTH in FORTH.*

Like many others, I joined the FORTH Interest Group after reading about FORTH in the August 1980 issue of *Byte*. I had become weary of programming in hand assembled machine language and it seemed to me that FORTH would be a big improvement. I have not been disappointed.

I was a bit dismayed, however, when I received my copy of 8080 fig-FORTH (1) to find that some modifications would be required to place the code in ROM. As my home-built S-100 system, which is a mixture of kits, assembled boards and wire-wrap, has no disk, and I was not willing to wait for the FORTH core to load from cassette every time I wanted to use it, I began at once to copy the listing in a form suitable for ROM.

I had a 16K EPROM board assigned to the upper quarter of memory and I was in the process of rewriting my utilities to use my new 1K memory mapped video board. I therefore decided to begin the ROM listing at the lowest address on the EPROM board, C000H (START) instead of 0000H. This meant transposing all the addresses up 48K, that is 0100H became C100H, 1000H became D000H, and so on. This is a relatively simple but time consuming task (especially without an assembler). It soon became apparent that more was required than simply transposing the addresses. Certain values change during the execution of FORTH and must be in RAM. Before proceeding very far with the project I found it necessary to assign functions to the various parts of my RAM. I have 16K of RAM in the first quarter of memory. I decided to use the first 10K for screens and the next 6K for RAM dictionary. The second quarter of the memory is not implemented in my system. The first kilobyte of the second half of the memory is occupied by my cassette interface. I then have 2K of RAM before my video display which begins at 8C00H. The rest of memory to C000H is not implemented.

I decided to use the 2K of RAM below the video board for the main FORTH stack, the terminal input buffer, return stack, user variables, a single 1K screen buffer and assorted patches. The size function and labels for these areas are well explained in the fig-FORTH installation manual (2). As there is plenty of room for these, I used larger sections of memory than suggested in the manual. Listing 1 is a set of equates which I used to replace those supplied on page 2 of the fig-FORTH 8080 Assembly Source Listing. If space is a problem, **US** and **RTS** could be restored to 64 and 160 respectively, and STACK could be reduced to 64.

LISTING 1

Revised Memory Allocation; Equates to establish stacks, terminal input buffer, user variables and a single 1K buffer.

```
0400 = KBBUF   EQU 1024            ; data bytes per buffer
00B4 = US      EQU 180             ; user variable space
00E0 = RTS     EQU 224             ; return stack and TIB
0060 = STACK   EQU 96              ; stack space
8400 = BTM     EQU 8400H           ; starting adr of 2K memory block
8460 = INITSO  EQU BTM+STACK       ; top of stack
8540 = INITRO  EQU INITSO+RTS      ; top of return stack
85F4 = BUF1    EQU INITRO+US       ; index of buffer
85F6 = BUFF    EQU BUF1+2          ; first data byte of buffer
89F8 = EM      EQU BUFF+KBBUF+2    ; upper limit of buffer
8A00 = STARTP  EQU EM+8            ; start of RAM patch
000A = NUMS    EQU 10              ; number of screens in system
1800 = LENGTD  EQU 1800H           ; length of RAM dictionary,6K
0000 = STARTS  EQU 0               ; start of first screen
2800 = LENGTS  EQU NUMS*KBBUF      ; length of screen memory,10K
2800 = STARTD  EQU STARTS+LENGTS   ; start of RAM dictionary
4000 = ENDD    EQU STARTD+LENGTD   ; end of dictionary area
8540 = USBASE  EQU INITRO          ; user variable base address
8572 = RPP     EQU USBASE+32H      ; return stack pointer
000B = BYTASK  EQU 11              ; number of bytes in "TASK"
280B = INITDP  EQU STARTD+BYTASK   ; initial value of dictionary pointer
8C00 = STARTV  EQU 8C00H           ; start adr of video display
```

LISTING 2

Patches to be moved to RAM on cold start. See pages 46 and 54 of Assembly Source Listing.

```
                               ORG C0E6H
C0E6 C5             PATCH      DB 0C5H        ;FORTH
C0E7 464F5254                  DB 'FORT'
C0EB C8                        DB 'H'+80H
C0EC B5CF                      DW VOCAB-13
C0EE CCCA           FORTH      DW DODOE
C0F0 E2CF                      DW DOVOC
C0F2 81                        DB 81H
C0F3 A0                        DB ' '+80H
C0F4 0028                      DW STARTD      ; start of RAM dictionary
C0F6 0000                      DW 0           ; end of vocabulary list
                             ; used by P@
C0F8 73             INPATCH    MOV M,E
C0F9 DB00                      IN 0
C0FB C9                        RET
                             ; used by P!
C0FC 7D             OUTPATCH   MOV A,L
C0FD D300                      OUT 0
C0FF C9                        RET
```

I used the first page of my FORTH ROM for machine code peculiar to my system. The first 15 bytes are vectors. (**JMP COLD JMP INSTATUS JMP KEYBOARD JMP VIDEODIS JMP PRINTOUT**) After these referenced machine code routines I have additional routines to move a block of memory (**MOVIT**), blank the front panel of my computer (**BLANK**), clear the video screen (**CLSCR**), and debug FORTH (**DEBUG**). **DEBUG** is a rewrite of the Debug Support on page 5 of the Assembly Source Listing. I moved it so there would be room for initial values of additional user variables. I added a new variable **BREAKPOINT** (857EH) to replace **BIP** in the listing so that **TNEXT LXI H, BIP** on page 5 becomes **DEBUG LXI H, BREAKPOINT**. Listing 2 consists of 3 patches to be moved into RAM (above limit 89F8H) on a cold start. These are required by certain FORTH words whose dictionary entries change during execution. (**FORTH P@ P!**)

This brings us to the start of the code on page 3 of the fig-FORTH listing. In addition to transposing the addresses, the following changes are required. Substitute **STARTD** for **TASK–7**. Since the word **FORTH** will be in RAM after start-up, beginning at **STARTP**, and since there are 8 bytes in the word before the label "FORTH," substitute STARTP+16 for FORTH+8 to initialize **VOC–LINK**. (In my system this is 8A00H + 10H = 8A10H.) Listing 3 is the list of additional user variables I added to the system.

The FORTH address interpreter is next. If you wish to use **BREAKPOINT** and **DEBUG**, substitute **JMP DEBUG** for the first three bytes of **NEXT**. I found this routine quite useful. Now that my fig-FORTH is operating, I suppose I could improve the speed of the system by removing this jump in an alternate EPROM chip, but I have not yet done so.

From this point on, if you have an assembler, your task is relatively simple, but if your reason for wanting your FORTH in ROM is your lack of resources, you may have to hand assemble the code as I did. You will soon get used to recognizing the addresses which have to be transposed and adding the extra ASCII characters not shown in the listing if a word has more than 6 characters. You have to watch for ocurrences of **RPP** (return stack pointer) and substitute your RAM address for the address given in the listing. (RPP = USBASE+32H) **RPP** is found in **(LOOP) (DO) I RP@ RP! ;S LEAVE >R R>** and **DOES>**. Be sure to transpose **UP** (pointer to user variable base address) from 0126H by

*Continued on next page*

```
                              LISTING 3
Initial values of additional user variables added to system. These are
loaded to USBASE + 32H = RPP on cold start.

C126 4085       UP      DW INITRO      ; init return stack pointer
C128 C03F               DW STARTV+3C0H ; init VP (1)
C12A 008C               DW STARTV      ; init VP1 (2)
C12C 008C               DW STARTV      ; init VP2
C12E 0100               DW 1           ; video display flag set
C130 0000               DW 0           ; printer flag clear
C132 0000               DW 0           ; init BREAKPOINT
C134 0000               DW 0           ; init XCUR (3)
C136 0000               DW 0           ; init YCUR (3)
C138 0028               DW STARTD      ; init HI
C13A 0000               DW 0           ; not implemented
C13C 0000               DW 0           ; not implemented
C13E 0000               DW 0           ; not implemented
C140 0000               DW 0           ; not implemented
C142 00                 DB 0           ; not implemented

(1) Initial value of VP ( video pointer ) is starting address of last
line of video display.
(2) Initial value of VP1 ( auxillary video pointer ) is starting address
of video display. VP2 is similar.
(3) Used by Editor.
```

**∗ Spectrum Model II Forth ∗**

Stand alone Forth System for the TRS Model II

- Superset of Forth 79
- Full CRT Support
- Interrupt driven KB, line printer and serial I/O drivers.
- High level redirectable character I/O
- Variable number block buffers employing least recently used scheme.
- Automatic disk format recognition.
- Extensions:
  - Z-80 Assembler
  - Strings
  - Array Words
  - Double Word Set
- Utilities:
  - Single Screen Editor
  - Multi-Screen Editor
  - Cross Reference Program with word search
  - Disk Formater (8 different formats)
    - 608 K capacity per drive
  - Smart Terminal program
  - Execution profile package
    - Word tracing with stack display

Single User System ............... 250.00
Manual ........................... 25.00

Run time Module licenses available upon request.

**∗ The-Data-Base ∗**

The-Data-Base is a menu driven, stand alone data filing and retrieval system for the TRS-80 Model II.

- User formated input screen to facilitate data entry.
- Query/Update module contains full edit features, including update, insert, delete and replace modes. Repeating feature of last record/field entered and default data key.
- Search feature allows comparisons of field constants, or field contents to another field within the same record.
- Disk spanning provides a maximum file capacity of over 2.2 megabytes on a four drive system.
- Ten user-defined keys provided
- Code file feature enables data to be entered in condensed and expanded modes, which then may be referenced globally by the system. Data may be retrieved in either condensed or expanded format.
- Merge/Copy feature provides for the appending of one file to another. Only matching fields of the files are copied, allowing the user to add or delete fields in the new file.
- Screen oriented Report Writer employs a virtual screen of 60 x 146 characters. Sorts may be performed on all fields in ascending or descending order, or any combination of same. Break field feature provided.

Arithmetic module enables the display of intermediate results.

- Simple password security provided for User, Update and System levels.

Single User System .............. 300.00
Manual .......................... 25.00
*Demo Disk and Manual .......... 50.00
*Full Credit applied to purchase of The-Data-Base.

Z-80 is a trademark of Zilog, Inc.
TRS-80 is a trademark of Tandy Corp.

SDS

**SPECTRUM DATA SYSTEMS**
5667 Phelps Luck Drive
Columbia, Maryland 21045
(301) 992-5635

(continued from page 17)

adding your offset (to C126H in my system) in **SP!** and **USER**.

When you reach the higher level FORTH words at page 21 in the Assembly Source Listing, note that labels which refer to other FORTH words are addresses which must be transposed to refer to ROM. A081 in **VOCABULARY** is *not* a label however, (it is a dummy header) so leave it as is. Leave the 18 bytes for the word FORTH blank as the word is part of our patch. Change the link field of **DEFINITIONS** from FORTH-8 (OFEA) to STARTP (8A00) to refer to FORTH in RAM. Similarly change the address for FORTH in **ABORT** to STARTP+8 (8A08).

The code for a warm start can be transposed to ROM in the usual manner, but I had to rewrite **CLD** and **COLD** to make the necessary changes for ROM. See Listing 4. Instead of using **CMOVE**, I used a short machine code subroutine in the first page of ROM. **(MOVIT: MOVE A,B ORA C RZ MOV A,M STAX D INX D INX D DCX B JMP MOVIT**)

If you wish to have complete messages instead of message numbers you will have to rewrite **MESSAGE** to print out text from ROM. For instance, if **TABLE** is the starting address of a table of addresses of the messages listed in order, then the sequence **TWO STAR LIT TABLE PLUS AT COUNT TYPE SPACE** could be inserted between **DDUP** and **MESS2** (replacing **ZBRAN** . . . **SPACE**). As this uses 4 bytes less you will have to adjust your branch offsets. I have not tried this yet.

The code for **P@** and **P!** must be changed to use the patches in Listing 2. Replace **LXI H, $+5 MOV M,E IN 0** in **P@** with **LXI H, STARTP+14H CALL STARTP+12H.** In **P!** replace **LXI H, $+7** with **LXI H, STARTP+18H** and replace **MOV A,L OUT 0** with **CALL STARTP+16H.**

This brings us to the CP/M Disk Interface on page 55 of the listing. I replaced this with a cassette interface. If you have copied out the listing this far, you should have little difficulty in constructing special FORTH words for your system. For example, Listing 5 is the word I added to turn my cassette recorder off. This is done in my system by sending 0 to port 4. Remember to

LISTING 4

```
Machine code for a cold start.

D08F 2A12C1      CLD      LHLD ORIG+12H  ; init stack pointer
D092 F9                   SPHL
D093 2112C1               LXI H, ORIG+12H; source       )   Initialize
D096 114685               LXI D, USBASE+6; destination )   low user
D099 011000               LXI B, 10H     ; length       )   variables
D09C CD89C0               CALL MOVIT     ; move         )
D09F 2126C1               LXI H, UP      ; source       )   Initialize
D0A2 117285               LXI D, RPP     ; destination )   high user
D0A5 011D00               LXI B, 1DH     ; length       )   variables
D0A8 CD89C0               CALL MOVIT     ; move         )
D0AB 21E6C0               LXI H, PATCH   ; source       )   Move
D0AE 11008A               LXI D, STARTP  ; destination )   patch
D0B1 011A00               LXI B, 1AH     ; length       )   to RAM
D0B4 CD89C0               CALL MOVIT     ; move         )
D0B7 2A38C1               LHLD RPP+12H   ; destination )   Move TASK
D0BA EB                   XCHG           ;              )   to start
D0BB 217DDA               LXI H, TASK-7  ; source       )   of RAM
D0BE 010B00               LXI B, BYTASK  ; length       )   dictionary
D0C1 CD89C0               CALL MOVIT     ; move         )
D0C4 CDA3C0               CALL BLANDCL   ; blank front panel, clear screen (1)
D0C7 01CDD0               LXI B, CLD1    ; load IP with pointer to CFA of COLD
D0CA C345C1               JMP NEXT       ; begin FORTH
D0CD D6C0        CLD1     DW COLD
D0CF 84                   DB 84H         ; COLD
D0D0 434F4C               DB 'COL'
D0D3 C4                   DB 'D'+80H
D0D4 82D0                 DW WARM-7
D0D6 11C6        COLD     DW DOCOL
D0D8 90D3                 DW MTBUF
D0DA DED0                 DW INITKEY     ; initialize keyboard
D0DC 58D0                 DW ABORT
D0DE E0D0        INITKEY  DW $+2         ; This is an alternate way
D0E0 DB07                 IN 7           ; to add machine code
D0E2 C345C1               JMP NEXT       ; words to your system.
(1) BLANDCL is a front panel blank and video screen clear peculiar to my
system. Write a suitable subroutine for your use.
```

LISTING 5

```
COFF: A FORTH word to turn off a cassette recorder by sending 0 to port 4.
Substitute the appropriate code for your system.

D2DC 84                   DB 84H         ; COFF
D2DD 434F46               DB 'COF'
D2E0 C6                   DB 'F'+80H
D2E1 CAD2                 DW 0D2CAH      ; NFA prevoius word
D2E3 E5D2        COFF     DW $+2
D2E5 AF                   XRA A
D2E6 D304                 OUT 4
D2E8 C345C1               JMP NEXT
```

insert the name field address of the preceding word at the link field of each word. Some of the disk interface words can be modified for use with a cassette system. Listing 6 shows some useful words. By this time the name fields and link fields should be obvious to you so I have listed only the labels, code fields and parameter fields. The label indicates the address of the code field of the corresponding FORTH word. **EMPTY–BUFFERS** can be used as is (transposed), but if you change its relative location be sure to change the reference in **WARM** and **COLD**.

The CP/M console and printer interface on page 63 of the listing can be used with a few changes. First, **IOS** is changed to **LXI H, START DAD D PCHL** (21 00 C0 19 E9). **PRINT** must be in RAM so change it to a user variable, **PF** (Print Flag) at RPP+OAH (857C). I modified **PCR** on page 64 to output only a **CR** instead of a **CR** and **LF** as I have the machine code for the video display written accordingly and the printer set accordingly. (**PCR: PUSH B MVI C,ODH MOV L,C CALL CPOUT POP B JMP NEXT**).

The rest of the listing is straightforward. For **BYE**, I use a jump to my utility directory instead of a jump to 0. If you add additional words after **TASK**, change the link field of **TASK** to the name field address of the last word

LISTING 6

```
Modified Disk Interface for use with cassette interface. Each block
corresponds to a full screen, but since block 0 is used to indicate terminal
operation, block 1 is for screen 0. ( that is, block# = screen# + 1 )

BLK# ( --- n ; Places the block number currently occupying the buffer onto the
stack. If the most significant bit is set it means block has been updated. )
BLOCKN: DOCOL FIRST AT SEMIS

UPDATE ( Modified to use BLK# )
UPDAT: DOCOL BLOCKN LIT 8000H ORR FIRST STORE SEMIS

BUF ( --- n ; A constant which leaves address of first data byte of buffer on
the stack.)
BUF: DOCON BUFF

HI ( --- n ; User variable containing address of end of screen area.)
HIGH: DOUSE 44H

R/W ( adr n f --- ; adr refers to buffer, n is block number, f is
flag,0=write,1=read. Disk primitive rewritten for simulation of disk by
memory. See fig-FORTH installation mamual. Error #6 is "SCREEN RANGE ERROR".)
RSLW: DOCOL TOR ONE SUBB BBUF STAR DUP HIGH AT GREAT LIT 6 QERR FROMR ZBRAN 4
SWAP BBUF CMOVE SEMIS

BUFFER ( n --- adr ; Assigns buffer to block n. If contents of bufer are
marked as updated, it is written to memory. Address left on stack is first
data byte of buffer.)
BUFFE: DOCOL BLOCKN ZLESS ZBRAN 10H BUF BLOCKN LIT 7FFH ANDD ZERO RSLW FIRST
STORE BUF SEMIS

BLOCK ( --- adr ; n is a block number, adr is the address of the first data
byte of the buffer. If the block is not in the buffer already, it is moved
there. If the block in the buffer is different and is updated it is first
moved to screen memory.)
BLOCK: DOCOL DUP BLOCKN SUBB DUP PLUS ZBRAN 10H DUP BUFFE SWAP ONE BSLW BRAN 4
DROP BUF SEMIS

FLUSH ( Write buffer to memory if updated.)
FLUSH: DOCOL ZERO BUFFE DROP SEMIS

LOAD ( n --- ; Begin interpretation of screen n.)
LOAD: DOCOL BLK AT TOR INN AT TOR ZERO INN STORE ONEP BLK STORE INTER FROMR
INN STORE FROMR BLK STORE SEMIS

( Labels not explained above will be found in the Assembly Source Listing.)
```

added and the link field of the first word added to the name field address of **.CPU** (DA58) and link the added words appropriately.

The revised listing is now ready for burning into EPROMs. Without an assembler, entering the program into memory is a tedious task but effort spent in checking at this stage will pay off later in debugging time. It can take quite a while to find that you have reversed the digits in a byte somewhere or have switched the bytes in a label. Check your IO routines in the first page of ROM carefully, be sure they work as intended and be sure your code for **COLD** is written and entered correctly.

If you read "8080 FIG-FORTH" on your screen on a jump from **START** (C000H), you are indeed fortunate. Your debugging procedure will depend on your front panel and utilities. If you can step your program to **JMP NEXT** in **CLD** (DOCA) and change the value of **BREAKPOINT** (at 857E) through the front panel you can run the program under control until it crashes and thereby locate the problem.

After I removed the bug from the system, I was ready to start writing programs in FORTH. I found *Starting FORTH* by Leo Brodie (3) very helpful. As I began to learn, I soon realized that I required an Editor. Fortunately, I was in contact with John Cassady (who implemented 8080 fig-FORTH) who supplied me with a compact Editor ideal for ROMing. I entered the screens in RAM using my "typewriter" utility and **LOAD**ed them. For a while I saved the screens on tape for use while I was placing the Editor in ROM. After loading the Editor, I printed out the RAM dictionary, identified the words by their ASCII characters and the **DOCOL** and **SEMIS** addresses and transposed the code to ROM, with the link field changes described above.

Implementing my fig-FORTH system has been a long laborious process but I have acquired a good understanding of the language as I have progressed and I am looking forward to increasing my programming skills. □

References:
1. *Fig-FORTH for 8080 Assembly Source Listing*. Release 1.1 September 1979. FORTH Interest Group.
2. *Fig-FORTH Installation Manual*. Release 1 by W. F. Ragsdale. November 1980. FORTH Interest Group.
3. *Starting FORTH* by Leo Brodie. Prentice-Hall, Inc.

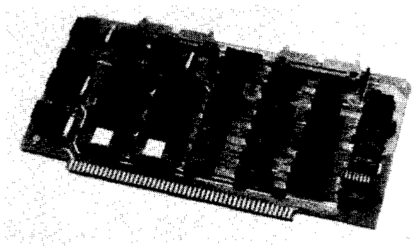
## DEVELOPMENT TOOLS

**Develop FORTH code for any target 8080/Z80 system on your current 8080/Z80 or Cromemco CDOS based system**

### 8080/Z80 METAFORTH CROSS-COMPILER

- Produces code that may be downloaded to any Z80 or 8080 processor
- Includes 8080 and Z80 assemblers
- Can produce code without headers and link words for up to 30% space savings
- Can produce ROMable code
- 79 Standard FORTH
- Price $450

**No downloading – No trial PROM burning. This port-addressed RAM on your S-100 host is the ROM of your target system**



### WORD/BYTE WIDE ROM SIMULATOR

- Simulates 16K bytes of memory (8K bytes for 2708 and 2758)
- Simulates 2708, 2758, 2516, 2716, 2532, 2732, 2564 and 2764 PROMS
- The simulated memory may be either byte or 16-bit word organized
- No S-100 memory is needed to hold ROM data
- Driver program verifies simulated PROM contents
- Price $495 each

## CONSULTING SERVICES

Inner Access provides you with Custom Software Design. We have supplied many clients with both Systems and Application Software tailored to their specific needs. Contact us for your special programming requirements.

## FORTH WORKSHOPS

**ONE-WEEK WORKSHOPS — ENROLLMENT LIMITED TO 8 STUDENTS**

**FORTH Fundamentals**

- Program Design
- Program Documentation
- FORTH Architecture
- FORTH Arithmetic
- Control Structures
- Input/Output
- The Vocabulary Mechanism
- Meta-Defining Words

**OCT. 4-8 NOV. 8-12**
**JAN. 3-7 FEB. 7-11**

$395 Incl. Text

**Advanced FORTH Applications**

- FORTH Tools
- Engineering Applications
- Floating Point
- Communications
- Sorting & Searching
- Project Accounting System
- Process Control
- Simulations

**NOV. 15-19**
**FEB. 14-18**

$495 Incl. Text

**Advanced FORTH Systems**

- FORTH Internals
- Assemblers and Editors
- Other Compilers
- Cross-Compilation Theory
- Romability, Multitasking, Timesharing
- File Systems/ Database Systems

**OCT. 11-15**
**JAN. 10-14**

$495 Incl. Text

Instructors: LEO BRODIE, GARY FEIERBACH and PAUL THOMAS
(For further information, please send for our complete FORTH Workshop Catalog.)

**Inner Access Corporation**

P.O. BOX 888 • BELMONT, CALIFORNIA 94002 • (415) 591-8295

BANKAMERICARD VISA
master charge

# Q T F
# Quick Text Formatter — Part II

*Leo Brodie*
*Chatsworth, California*

In the last issue of *FORTH Dimensions*, I introduced my Quick Text Formatter, a very simple but powerful text processor written in FORTH. That issue included the code for the formatter section of the application. This article continues with the QTF Editor.

The QTF Editor allows you to edit FORTH blocks which will serve as source blocks for the formatter. The Editor is designed with text manipulation in mind, and offers cursor-control, wrap-around, insert, delete, replace and string-move functions.

**I/O Option**

I've included two versions of display formatting for you to choose from. In the first approach, called Continuous Refresh, if you insert text in the middle of a line, the remainder of the line visibly shifts across the screen. At a certain point, the word or words at the end of the line drop down to the beginning of the next line, and the remainder of the block is adjusted as needed. This approach is fine if you're a single user on a system and you have fast video I/O.

You may need the alternate approach, which I call the Burst I/O, if you're working on a multi-tasked system with other users or with a slow baud rate. In the Burst I/O approach, *as soon as you begin to enter text*, the remainder of the screen past the cursor is blanked, so that you can type into the space without the remainder of the text having to be continually refreshed. A copy of the remaining text appears near the bottom of the screen, so you can see what will follow the point of insertion.

When you press return, the remaining text reappears, following the inserted text.

Another reason to use the second approach is if your system does not include the word **<CMOVE** and you're not up for writing it in code. I'm afraid you really need a fast version of **<CMOVE** for satisfactory results.

Similarly, there are two kinds of text deletion. For Burst I/O, you position the cursor at the beginning of the string to be deleted, then type "X"s over the full length of the string. (This is in "Command Mode," so the "X"s are commands, not characters.) When you press the return key, the X'ed-out text is deleted. Again, this avoids continuallly refreshing the remainder of the screen.

(A nice side-effect is that if you overshoot the number of "X"s, you can backspace and the former characters will reappear, since nothing is actually clobbered in memory until you press "return.")

If you're using the Continuous Refresh version of insertion, then you can also delete text by being in Enter Mode, positioning the cursor at the end of the text to be deleted, then pressing the backspace key until the string has been gobbled up by the text to the right of the cursor. You still have the "X"-ing command available though, if you prefer it.

**Another Unusual Approach**

One of the problems of using someone else's software is that you have to put up with that person's quirky preferences, at least until you understand the application well enough to change it. In the case of the QTF Editor's cursor commands, I've implemented a quirky preference of my own, which I'll explain right now in case you don't like it.

Most Editors of this type employ control codes to indicate cursor positioning commands. You have to hold down the control key while you press various other keys to move the cursor up, down, left, or right, in order to let the Editor distinguish between commands and characters.

My approach, though, was to use "modes." When you first enter the Editor, you are in "Command Mode." Now various keys represent commands, including both cursor movement commands as well as commands to enter other modes.

As for my choice of keys to control the cursor with, I prefer to use the position of the keys on the keyboard rather than an association of actual letters. Instead of "U" for "up," etc., I picked four keys that were adjacent and nearest the first two fingers of my right hand:

```
          up
           i
  left  j     k  right
           m
          down
```

Even with my IBM Personal Computer's special set of cursor keys at the right side of the board, I prefer the mode approach because it takes less hand movement. But, as I say, that may just be my own quirk. You can easily change the keys by modifying the key function table in screens 32 and 33, which I'll discuss later.

For rapid horizontal cursor moves (four characters at a time), just capitalize J and K.

**Prerequisites**

An early warning: to make this thing work, your terminal must be able to perform certain functions. In this application they are named:

| | |
|---|---|
| **x y XY** | moves the cursor to coordinates x and y, where 0,0 is the upper left. |
| **CLR−>LN** | clears the line to the right of the cursor. |
| **CLR−>SCR** | clears the screen to the right of and below the cursor. |
| **PAGE** | clears the entire screen; homes the cursor. |

You'll have to write these routines if they don't already exist in your system. Redefine your routines to bear the names in Block 18, or simply patch their addresses into the execution

*Continued on next page*

**QTF** (continued from page 21)

variables. (I used execution variables just to avoid confusing things).

### Using the Commands

To enter the Editor, type the word
**n write**
where "n" is the number of the block you want to edit. If there's anything in the block already, it will be typed out. Since our formatter application requires a continuous stream of text within each block (no embedded carriage returns), the Editor decides where to end each line on the display — actually, the first occurrence of a blank after the 55th character on each line. (Remember, the Editor formats the contents of the screen differently than the formatter will display your text.)

The cursor will appear just after the last character displayed, or, if the block is empty, at the home position. The number of characters in the block (the current "length") will appear in the lower right hand corner.

You can use the cursor movement keys to position the cursor to any character that actually exists in the block. You can move it backwards or up, but not forward or down (beyond the "length"), until you enter more text. If you move the cursor all the way back to the beginning of a line, it will then skip up to the end of the next line above — not to the 79th column, but to the blank space that follows the last word on the line. And you can only move the cursor forward as far as the last character in each line (the space), then it will drop to the beginning of the next line.

While in command mode you can also get into Enter Mode, Replace Mode, or Delete Mode by pressing "e," "r," and "d" respectively.

Table 1 shows all the commands that are available as keystrokes in Command Mode.

### Enter Mode

Press the "e" key while in Command Mode to begin entering text. The words "ENTERING MODE" will appear in the upper right-hand corner. In this mode, all the alphanumeric keys on your keyboard will be considered characters you want to enter, not commands.

I've already discussed the Editor's two approaches to the I/O problem in Enter mode. But with either approach the Editor's wrap-around strategy applies. This means you don't press "return" at the end of each line; you just keep typing and the editor will worry about line breaks. Press return only when you want to get back to Command Mode.

If you make a mistake while in Enter Mode, press the backspace key. You can even backspace past the beginning of a displayed line and up to the end of the previous line, just like with cursor movements.

In either the Burst or Continuous versions of Enter Mode, you can backspace all the way back to the beginning of the block, thereby deleting all such text.

As you enter text, make sure the length in the lower right hand corner doesn't exceed 1024, or you'll lose the end of your text. I recommend that you only fill blocks about 2/3 full, to leave room for later insertions.

### Delete Mode

I've also covered Delete Mode in the discussion on the I/O alternatives earlier. For either version, just "X" over the string to be deleted. Pressing the return key joins the parts of the text on either side of the deletion.

While in Delete Mode, the only valid keystrokes are "x," upper case "X" for fast deletions, backspace and return. Any other key will emit a beep.

A quick way to clear everything from the cursor to the end of the screen is the "cntrl-c" key. Since this is a dangerous command, it requires holding down the "control" key and the "c" simultaneously. You can clear the entire contents of a screen with the combination "a" "cntrl-c" (the "a" key homes the cursor to the beginning of the screen).

### Replace Mode

Use this mode when you want to correct text without moving any text forward (as in insert) or backward (as in delete). For example, to change the word "most" to "many," simply move the cursor to the "o," then enter Replace Mode by typing "r." Then type "any." Be sure to hit "return" when you're done replacing!

### The "Take" Command

The "t" (for Take) key takes the string you just "x"ed out, and inserts it wherever the cursor is now. In combination with "x," you can use this command to move any length of text from one place to another, even from one block to another. Use it to swap phrases, sentences, paragraphs, etc.

You can also use "t" if you need multiple copies of the same string, without having to retype it each time. Just "x" out the string, then press "t" to take it back, then press "t" again to take a second copy, and again as many times as you need. This technique works because "x" saves the text that you delete in a buffer. The "t" command copies whatever is in that buffer into your block.

If you need to transfer a LOT of text from the end of one block to another block, or even copy an entire block,

**Table 2 Editor-related FORTH commands**

| | |
|---|---|
| **w** | re-enters the Editor, using the block most recently displayed |
| **n** | re-enters the Editor, using the next block |
| **b** | re-enters the Editor, using one block back |
| **lb** | enters the Editor, displaying your document's loadblock. When you press return to exit, **lb** returns to the block you were originally editing. You must first identify your loadblock with the phrase **n loadblock !** where "n" is the number of your loadblock. |

you can use "t" in combination with "cntrl-c." "Cntrl-c" saves what it deletes in the same buffer that "x" does. For example, to transfer the second half of one screen to another screen, first position the cursor at the start of the text to be transferred, press "cntrl-c,", move to the other screen, then press "t."

### Extensions

As we mentioned earlier, the word **write** gets you into the Editor. There are also a few variations on that word for your convenience. These are summarized in Table 2.

Another optional word, which I include, paren'ed out, in block 35, is called **catch**. Catch is only needed when the Burst I/O version of Enter Mode is used. Because in this version you can't see how much text lies below the cursor, **catch** is your protection in case you insert too much and lose some text off the "bottom" of the block. If this happens, you can "catch" everything that existed to the right of the cursor at the time you began "Enter Mode" in another block. For instance,

**55 catch**

copies all such text to block 55, and enters the Editor.

### Theory of Operation

One of the reasons I'm publishing this application is the need for good, exemplary FORTH programs for teaching and learning. I've done my best to make this code elegant and readable, and I think it reflects a good design since, to be honest, it's been through quite a few design iterations as I've expanded its capabilities over the past six months.

With this in mind, I have an obligation to explain some of the design. Or to put it more accurately (since every decent programmer loves to brag about his code), I now have an excuse to explain some of the design. I'll see how much I can explain before this article gets too long.

### Two Dimensional Execution Table

At the heart of the key interpreter lies an execution table defined in blocks 32 and 33, called **FUNCTIONS**. When a key is struck, its value is compared against the ASCII values that appear in column one of the table. If a match is found, then the appropriate function from one of the other columns is executed. For example, suppose an "e" is pressed. If the Editor is in Enter Mode, **ICHAR** will be executed. **ICHAR** inserts a character, in this case the "e." In Replace Mode, **RCHAR** will be executed, to replace a character. In Delete Mode, the "e" is invalid, so a beep will sound. (I renamed the usual BELL function as ---- for clarity in this table.) In Command Mode, however, an "e" will execute **ENTER** (block 29), which sets the Editor to Enter Mode.

If no match is found, then the appropriate function from the last row of the table is executed. For example, a "q" will be inserted or replaced while in Enter or Replace Modes, but will cause a beep while in Delete or Command Modes.

The word **FUNCTION** in block 34 is doing the work. First it finds the address in the **FUNCTIONS** table where the key match occurred, then it offsets this address by the contents of the variable **MODE** (which contains a 2, 4, 6, or 8). It then fetches the address of the appropriate function from the table and **EXECUTES** it. The modes are defined as **CONSTANT**s in Block 19, so the phrase

**DELETING MODE !**

for example, sets **MODE** to 6.

An alternative approach might have been to design key-interpreters within the key-interpreter. For instance, pressing "e" in Command Mode might have executed an inner loop which also executed **KEY** and inserted each character. This inner loop, however, must also detect backspaces and carriage returns. Similarly, Replace Mode and Delete Modes would have needed their own inner key-interpreters as well. I think the result would hae been much more cumbersome.

From what I understand, this implementation of a two-dimensional vectored execution table runs along the lines of classical State Machine theory. Whatever, I found this approach remarkably easy to work with, due to the direct correspondence between what I was trying to do and the notation I used to describe it.

By the way, the design is such that you can add a new key and row of commands without having to change any code elsewhere. This works because the address of the end of the table is given by the constant **FUNCTIONS>**, defined as **HERE** immediately after the table is compiled.

The functions associated with Delete Mode, Replace Mode, and Enter Mode are implemented in Blocks 27, 28 and 29 respectively.

### Data Structure

The design of this Editor is more complicated than that of ordinarly FORTH screen editors because of the uneven line lengths. For instance, moving the cursor up one line isn't just a matter of moving back exactly 64 characters; the exact number of characters between two vertically adjacent characters depends on the number of characters displayed in the higher line.

As the user moves the cursor within the horizontal and vertical frame, the Editor must always keep track of its analogous position within the 1024-byte array (the conventional **R#**).

To correlate the two points of view, I implemented a table called **EDGES**. For each line of text on the display, there are a pair of values in the table. One represents the x-cursor position of the last character displayed on the line (actually the space); the other indicates the value of **R#** for that same character.

For example, if there are 57 characters displayed on the 0th line, the final character is the 57th. The first two entries of the table, then, are both 57. If there are 60 characters on the next line, then the final x-cursor position for that line is 60, and the value of **R#** for that final character is 118 (the first character on that line is **R#** 58, so 58 + 60 = 118.

Say there are only three lines, and the final line contains 10 characters. Thus the first table entry for that line is 10, the other entry is 129. All entries

*Continued on next page*

for subsequent lines will contain zeros.

We can picture the table like this:

|  |  |
|---|---|
| **57** | **57** |
| **60** | **118** |
| **10** | **129** |
| **0** | **0** |
| **0** | **0** |
|  | **etc.** |

(As a debugging tool, the word **SNAP** in block 36 displays the values of the **EDGES** table, as well as several of the other major variables. If you want to use it, load block 36 from line 10 of block 34, and leave the word **SNAP** in the definition of **(WRITE)** on line 13. Now you can execute the word **W/SNAP** to turn **SNAP** on, or **WO/SNAP** to turn it off.)

The phrase **EDGE** @ fetches the x-cursor value for the current line from the table. For example, if we're currently in the middle line, **EDGE** @ will return 60. The phrase **EDGE 2+** @ fetches the associated **R#** value for the current line, e.g., 118.

The variables **XCUR** and **YCUR** contain the current coordinates of the cursor, relative to the text display area (**YCUR** is 0 for line zero of the text; this is actually line 2 on your terminal).

Here is the algorithm for moving the cursor from the middle of one line down to the next line:

**1 YCUR +!**
(moves the cursor to the next line down)

**EDGE** @ **XCUR** @ –
(computes the distance between where the cursor is horizontally on the new line, and the last character of that line, as a positive number.)

**EDGE 2+** @ **SWAP** –
(subtracts this value from the **R#**-value of the last character on this line.)

**R# !**
(having computed the value of **R#** associated with the character the cursor has just landed on, stores that value into **R#**.)

The algorithm is complicated by the fact that the space below the current position might not be a valid character position. For example, if the cursor sits at the 60th character in the middle line, and the user tries to move it down, the cursor should slide down and left to the last character position of the next line, the 10th position.

Furthermore, it turns out that this algorithm is the same for both moving down and for moving up, the only thing that changes being the value (1 or −1) that is plus-stored into **YCUR**. That being the case, the complete algorithm is factored as the word **UP/DOWN** in block 22. Enough said.

I haven't begun to discuss the commands that format and display the text, and keep the display looking right during the editing process. A particularly interesting routine is the one which minimizes the amount of screen typing necessary after a short insert or delete, by only typing as many lines as are necessary to take up the slack. But this article is already too long, so I guess I'll just bag it. I hope I've given you enough hints to pursue it further.

**If You Type It In . . .**

As with the Formatter, the Editor is written according to 79-Standard, ala *Starting FORTH*. If you're running a FIG system, then preload the following definitions:

```
: VARIABLE   0 VARIABLE ;
: CREATE   VARIABLE -2 ALLOT ;
: WORD   WORD HERE ;
: >IN   IN ;
: ?DUP   -DUP ;
: NOT   0= ;
```

Make sure you set the value of the constant **RETURN** to 13 or 0, depending upon which number **KEY** returns when you press carriage return. This constant is in block 18.

A related problem will be your definition of ', ("tick-comma") which is used to create the **FUNCTION** table. The purpose of this word is simply to find the address of the next word in the input stream, and comma into the dictionary whatever address (cfa or pfa) is appropriate for your system's version of **EXECUTE**. Three versions — FIG, 79-Standard, and Starting FORTH are provided in Block 18.

Another precaution: the word **ASCII** is normally used inside colon definitions to compile the next word in the definition as a literal. I'm using it in the creation of the **FUNCTIONS** table, and this is not inside a colon definition. In this case, **ASCII** should simply return the value of the character on the stack. Make sure that you have a definition of **ASCII** that does this, or else just comma in the actual numbers (no fun, I know).

And still another precaution: some FORTH systems store a null into the first cell of empty blocks. If you insert text into such a screen with this Editor, those nulls will ride along, glued to the last character you type. This may cause funny problems at format time. Probably the best solution is to add a routine to the Editor which scans the block for control characters and changes any to blanks.

Finally, I've used the words **T** and **F** to indicate true and false flags, respectively. This is purely in the interest of style, since it often helps to know whether 1s and 0s are to be treated as numbers or as flags. Their definitions are, of course:

```
0 CONSTANT F
1 CONSTANT T
```

*Leo Brodie is an author, lecturer and consultant on FORTH programming. He is currently working on his second book, and is employed as a consultant to IBM in San Jose.*

# Quick Text Formatter
## Editor Glossary

### Command Mode
In this mode, the following keys perform these functions:

| | |
|---|---|
| i | moves cursor up |
| j | moves cursor left |
| k | moves cursor right |
| m | moves cursor down |
| J | moves cursor left four characters |
| K | moves cursor right four characters |
| a | sets cursor to first character in block |
| z | sets cursor to last character in block |
| ctrl-c | Cuts-off everything to the right of the cursor. |
| n | displays Next block |
| b | displays one block Back |
| e | enters "Enter Mode", allowing you to enter or insert text. |
| r | enters "Replace Mode", allowing you to overwrite text. |
| x | deletes a character, and enters "Delete Mode". |
| X | deletes four characters at a time. |
| t | Takes the text that was most recently deleted with "x" or "cntrl-c", and inserts it at the current cursor position. |
| return | returns to FORTH |

```
Screen # 18          crc ver = 8364
( Quick Text Formatter          By Leo Brodie        09/25/82 )
   ( System dependent words)
VARIABLE 'PAGE              VARIABLE 'CLR->LN
VARIABLE 'CLR->SCR          VARIABLE 'XY
: PAGE    'PAGE @ EXECUTE ;
: CLR->LN    'CLR->LN @ EXECUTE ;
: CLR->SCR   'CLR->SCR @ EXECUTE ;
: XY    'XY @ EXECUTE ;

: BOTTOM    0 20 XY  CLR->LN ;
: .MODE    65 1 XY  CLR->LN ;
13 CONSTANT RETURN  ( value KEY returns when <return> pressed)
: ',    [COMPILE] '  , ;      ( 79 Standard)
( : ',    [COMPILE] '  CFA , ;  ( fig VERSION)
( : ',     '  , ;              ( Starting FORTH)
-->
```

```
Screen # 19          crc ver = 31127
   ( VARIABLES & CONSTANTS                                07/28/82 )
78 CONSTANT WIDE           55 CONSTANT NEAR-RIGHT

 2 CONSTANT ENTERING       4 CONSTANT REPLACING
 6 CONSTANT DELETING       8 CONSTANT COMMAND
VARIABLE MODE  ( command, enter, replace or delete)
 4 2* CONSTANT #MODES ( number of modes, in bytes)
32767 CONSTANT MOOT ( impossible # of shifts to accomodate)
VARIABLE XCUR              VARIABLE YCUR
VARIABLE TALLY  ( count of chars entered)
19 CONSTANT #LINES
CREATE EDGES  4 #LINES 1+ * ALLOT  ( table of right edges: )
   ( xcursor-pos, r#, for each line.                      )
-->
```

10/13/82

```
Screen # 20          crc ver = 3403
   ( VARIABLES & CONSTANTS                            07/28/82 )
CREATE xHOLDING  1026 ALLOT ( delete buffer; cell 0 = count)
VARIABLE #KEY  ( latest key entered)
VARIABLE ?ESC  ( true for escape from outer loop)
CREATE OLDPOS  6 ALLOT  ( ycurs xcurs r#)
VARIABLE REALIGN  ( true = need to realign in replace mode)
VARIABLE DONE  ( true = stop REWRITE)
VARIABLE SHIFTS  ( # to adjust table r#s in SET-REST)
VARIABLE LENGTH  ( # of writeable chars.)
VARIABLE RIGHT  ( current right margin)
-->
```

```
Screen # 21          crc ver = 34110
   ( Precursors                                       07/28/82 )
: BUFF ( -- adr: beg. of buffer)  SCR @ BLOCK ;
: CHARACTER ( -- adr: current pos. in buffer)  BUFF  R# @  + ;
: AT   XCUR @  YCUR @ 2+  XY ;
: .CHAR   AT  CHARACTER C@  EMIT ;
: ROOM ( -- n)  1024  R# @ - ;
: !OLDPOS ( save current cursor position)
  XCUR @  YCUR @  OLDPOS 2!  R# @ OLDPOS 4 + ! ;
: @OLDPOS ( restore previous cursor position)
  OLDPOS DUP  2@ YCUR ! XCUR !  4 + @ R# ! ;
: COMMAND!   .MODE  ." Command mode"  COMMAND MODE !  AT ;
: RET  ( drop to beg. of next line; let RCHAR know)
  AT CLR->LN  1 YCUR +!  0 XCUR !  T REALIGN ! ;
 -->
```

```
Screen # 22          crc ver = 20164
   ( Cursor movements                                 07/30/82 )
: EDGE ( -- adr)  YCUR @  2* 2*  EDGES + ;
( These words conform to preset edge table:)
: FORWARD   R# @  LENGTH @ < IF  1 XCUR +!
   XCUR @  EDGE @ >  IF  RET  THEN  1 R# +!   THEN ;
: BACKWARD   R# @ IF  XCUR @  0= IF  -1 YCUR +!
   EDGE @  XCUR !  ELSE  -1 XCUR +!  THEN
   -1 R# +!  THEN ;
: UP/DOWN ( n)  YCUR +!  EDGE 2@ ( r# x)  XCUR @ -  DUP
   0< IF  DROP  EDGE @ XCUR !  ELSE  -  THEN  R# ! ;
: UP   YCUR @ 0 > IF  -1 UP/DOWN  THEN ;
: DOWN   EDGE 4 + @ IF  1 UP/DOWN  THEN ;

-->
```

```
Screen # 23          crc ver = 45597
   ( Maintain table of right edges                    07/30/82 )
: SET ( set both edge markers for this line in table; and
   flag REWRITE to stop if shifts have been accomodated)
   R# @ 1-  XCUR @ 1-  DUP EDGE @ -   SHIFTS +!
   SHIFTS @ 0= DONE !   EDGE 2! ;
: FIT  ( go to next line, if necessary)
   RIGHT @  XCUR @ < IF  SET  RET  THEN ;
: ECHO ( c)  ( display c; break line on blank or end of line)
   DUP AT EMIT  1 XCUR +!  1 R# +!  BL =  XCUR @ WIDE =  OR
   IF  FIT  THEN ;
: ADJUST ( adjust r#'s in edge-table after x's or t's)
   YCUR @  #LINES OVER DO  I YCUR !  EDGE @ IF  EDGE 2+ @
   TALLY @ -  1024 MIN  EDGE 2+ !  THEN  LOOP  YCUR ! ;
-->
```

```
Screen # 24          crc ver = 20218
  ( Maintain table of right edges                     07/30/82 )
: NO-MORE  ( zero-out edge table from next line on down)
  YCUR @  #LINES OVER 1+  DO  I YCUR !
  0 0 EDGE 2!  LOOP  YCUR ! ;
: PATCH    ( fix remainder of edge table, after actual REWRITE)
  DONE @ IF  ADJUST  ELSE  R# @ XCUR @  EDGE 2!
  NO-MORE  CLR->SCR  THEN ;
: CHANGE  ( n)  ( pos. or neg. change in # characters)
  DUP TALLY !  DUP SHIFTS !  NEGATE
  LENGTH @ +  1024 2DUP MIN  LENGTH !  /  MOOT !  SHIFTS +! ;
      ( if length exceeds 1024, shifts are "moot")
-->
```

```
Screen # 25          crc ver = 16193
  ( REWRITE                                           07/30/82 )
: REWRITE   ( display remainder of text)
   !OLDPOS  F DONE !  LENGTH @  DUP IF  R# @ -  ?DUP IF
      0 DO  NEAR-RIGHT XCUR @ -  DUP 0>  IF
           LENGTH @  R# @ -  MIN  AT CHARACTER  OVER TYPE
           DUP XCUR +!  DUP R# +!
           ELSE  DROP CHARACTER C@ ECHO  1 THEN  DONE @
       IF LEAVE THEN  +LOOP  THEN
   ELSE DROP .CHAR THEN  PATCH ;
-->
```

10/13/82

```
Screen # 26          crc ver = 20392
   ( Various Modes Setup                              08/02/82 )
: REATTACH ( source-buffer-adr)
   DUP @ IF  .MODE  AT  DUP 2+ CHARACTER  ROT @ ROOM MIN  CMOVE
   REWRITE  @OLDPOS AT  ELSE DROP  THEN  COMMAND! ;
: HOME   0 R# !   0 XCUR !  0 YCUR ! ;
: ---   7 EMIT ( bell) .CHAR ;
: ESCAPE ( leave outer loop)  T ?ESC ! ;
-->
```

```
Screen # 27          crc ver = 11626
    ( Delete Mode                                     07/30/82 )
: XCHAR  ( delete one char.)
   R# @ LENGTH @ < IF  AT  ASCII X EMIT  FORWARD  THEN ;
: X-ING  ( begin delete mode)  !OLDPOS  R# @ TALLY !  XCHAR
   DELETING MODE !  .MODE  ." DELETE MODE"  ;
: CLOSE-UP   CHARACTER DUP  TALLY @ -
    DUP xHOLDING 2+  TALLY @  DUP xHOLDING !  CMOVE
    ( save deleted)   ROOM  CMOVE  ( close gap) ;
: BLANK->END   ROOM @OLDPOS  CHARACTER +  TALLY @  BL FILL ;
: xSTOP ( exit delete mode)
   R# @  TALLY @ -  CHANGE  CLOSE-UP  BLANK->END  AT SPACE
   REWRITE @OLDPOS  COMMAND! ;
: x<  ( delete mode backspace)  TALLY @ R# @ - IF
   BACKWARD  .CHAR AT  ELSE  T DONE !  xSTOP  BACKWARD  THEN ;
-->
```

```
Screen # 28          crc ver = 28142
   ( Replace Mode                                     08/02/82 )
: REPLACE ( go into replace mode)   REPLACING MODE !  0 TALLY !
   F REALIGN !  .MODE ." REPLACE" ;
: RCHAR  ( replace one char.)
   R# @ LENGTH @ < IF  #KEY @ DUP CHARACTER C!  ECHO
   REALIGN @ IF  REWRITE @OLDPOS AT  F REALIGN !  THEN  THEN ;
: r< ( replace-mode backspace)  BACKWARD  .CHAR ;
: rSTOP ( exit replace)
   0 SHIFTS !   REWRITE @OLDPOS   COMMAND! ;

  -->  ( paren out this arrow to load Burst I/O version)

37 LOAD ( Burst I/O version)
```

```
Screen # 29          crc ver = 2278
    ( Enter mode  CONTINUOUS I/O VERSION              09/30/82 )
: ENTER   ( go into Enter )
   R# @ 1024 < IF ENTERING MODE !  .MODE ." ENTERING MODE"
   0 TALLY !  THEN ;
: PATIENT   RIGHT @  1+  77 MIN  RIGHT !  EDGE @ 75 =
   XCUR @ NEAR-RIGHT 5 + =  OR  IF  NEAR-RIGHT RIGHT !  THEN ;
: ICHAR ( insert one char.)   R# @   1024 < IF  CHARACTER DUP 1+
   LENGTH @ 1023 MIN R# @ - <CMOVE  #KEY @ DUP CHARACTER C!
   -1 CHANGE  ECHO  R# @ LENGTH @ -  IF PATIENT REWRITE @OLDPOS
   THEN  THEN ;
: eSTOP  NEAR-RIGHT RIGHT !  0 CHANGE   REWRITE @OLDPOS
   COMMAND! ;
: e< ( backspace in Enter mode)   RIGHT @ 1-  NEAR-RIGHT MAX
   RIGHT !  R# @ IF  BACKWARD  AT SPACE  CHARACTER DUP 1+ SWAP
   LENGTH @  R# @ - CMOVE  1 CHANGE  REWRITE @OLDPOS  THEN ;
-->
```

```
Screen # 30          crc ver = 29713
   ( Misc. functions                                  07/30/82 )
: TAKE  ( spread radr. of block; restore xHOLDING)
   CHARACTER ( source) DUP xHOLDING @ + ( dest)
   ROOM  xHOLDING @ - ( count) DUP 0> IF <CMOVE ( spread)
      ELSE  2DROP DROP THEN
   xHOLDING @ NEGATE CHANGE  xHOLDING REATTACH ;
: -HOME ( move to last non-blank char.)
   LENGTH @ DUP R# !  #LINES 0 DO I YCUR ! DUP EDGE 2+ @ = IF
   LEAVE  THEN  LOOP DROP EDGE @  XCUR ! ;
: CUTOFF ( erase to end)
   CHARACTER xHOLDING 2+  LENGTH @ R# @ -  DUP xHOLDING !
   CMOVE  CHARACTER ROOM  BL FILL CLR->SCR
   COMMAND!  R# @  DUP LENGTH !  XCUR @  EDGE 2!  NO-MORE ;
-->
```

```
Screen # 31          crc ver = 29467
( Multiple-operation keys:                            10/13/82 )
4 CONSTANT MULTIPLE
: BACKS     MULTIPLE 0 DO BACKWARD LOOP ;
: FORWARDS   MULTIPLE 0 DO FORWARD  LOOP ;
: XCHARS    MULTIPLE 0 DO XCHAR    LOOP ;
: NEXTSCR    1 SCR +!  ESCAPE ;
: BACKSCR   -1 SCR +!  ESCAPE ;
-->
```

10/13/82

```
Screen # 32          crc ver = 49176
   ( Function matrix                                    08/09/82 )
CREATE FUNCTIONS
              ( E-mode     R-mode     X-mode      Command)
 ASCII e ,    ', ICHAR    ', RCHAR    ', ---     ', ENTER
 ASCII x ,    ', ICHAR    ', RCHAR    ', XCHAR   ', X-ING
 ASCII i ,    ', ICHAR    ', RCHAR    ', ---     ', UP
 ASCII m ,    ', ICHAR    ', RCHAR    ', ---     ', DOWN
 ASCII j ,    ', ICHAR    ', RCHAR    ', ---     ', BACKWARD
 ASCII k ,    ', ICHAR    ', RCHAR    ', ---     ', FORWARD
 ASCII r ,    ', ICHAR    ', RCHAR    ', ---     ', REPLACE
 3 ( ^c) ,    ', ---      ', ---      ', ---     ', CUTOFF
 8 ( del) ,   ', e<       ', r<       ', x<      ', BACKWARD
 ASCII a ,    ', ICHAR    ', RCHAR    ', ---     ', HOME
 ASCII z ,    ', ICHAR    ', RCHAR    ', ---     ', -HOME
-->
```

```
Screen # 33          crc ver = 8347
 ( Function matrix cont'd                                 08/09/82 )
  ASCII J ,     ', ICHAR     ', RCHAR    ', ---      ', BACKS
  ASCII K ,     ', ICHAR     ', RCHAR    ', ---      ', FORWARDS
  ASCII n ,     ', ICHAR     ', RCHAR    ', ---      ', NEXTSCR
  ASCII b ,     ', ICHAR     ', RCHAR    ', ---      ', BACKSCR
  ASCII t ,     ', ICHAR     ', RCHAR    ', ---      ', TAKE
  ASCII X ,     ', ICHAR     ', RCHAR    ', XCHARS   ', X-ING
  RETURN ,      ', eSTOP     ', rSTOP    ', xSTOP    ', ESCAPE
 ( other) 0 ,   ', ICHAR     ', RCHAR    ', ---      ', ---
HERE CONSTANT FUNCTIONS>  ( end of FUNCTION table)

 -->
```

```
Screen # 34          crc ver = 60149
( QTF Editor                                            08/09/82 )
#MODES 2+ CONSTANT GROUP ( bytes in FUNCTIONS for each command)
FUNCTIONS> GROUP - CONSTANT 'NOMATCH
: FUNCTION  ( key)   'NOMATCH  SWAP
   FUNCTIONS> FUNCTIONS DO  DUP I @ = IF 2DROP  I 0 LEAVE
   ( replace nomatch adr w/ match)  THEN  GROUP  +LOOP  DROP
   ( adr in table -- )  MODE @ +  @  EXECUTE ;
: INIT   MOOT SHIFTS !  BUFF 1024 -TRAILING LENGTH !  DROP
   HOME  NEAR-RIGHT RIGHT !  COMMAND! ;
: STAGE  ( scr)  PAGE  DUP .  SCR !   INIT ;
( 36 LOAD   ( snapshot debugger)
: (WRITE)  ( scr)  STAGE  REWRITE  NO-MORE  F ?ESC !
   BEGIN  75 19 XY  LENGTH @  4 .R  AT  KEY DUP  #KEY !
   FUNCTION  ( SNAP ) ?ESC @ UNTIL  UPDATE  BOTTOM  CR ;
-->
```

```
Screen # 35          crc ver = 61269
( QTF Editor                                            08/09/82 )
: write ( scr)   ( enter editor)
    BEGIN  DUP (WRITE)  SCR @ - ( n or b) WHILE
    SCR @ REPEAT  ( ." Saving on disk "  FLUSH  (optional ) ;
: w  ( reenter editor)   SCR @ write ;
: n    SCR @ 1+ write ;
: b    SCR @ 1- write ;
: index   INDEX ;
VARIABLE loadblock
: lb   SCR @  loadblock @ (WRITE)  write ;
17 LOAD  ( terminal-specific commands -- write your own)

( : catch  ( scr )  DUP  SCR !  INIT  CUTUFF  HOLDING 2+
    SWAP BLOCK  HOLDING @ CMOVE  w ;)
```

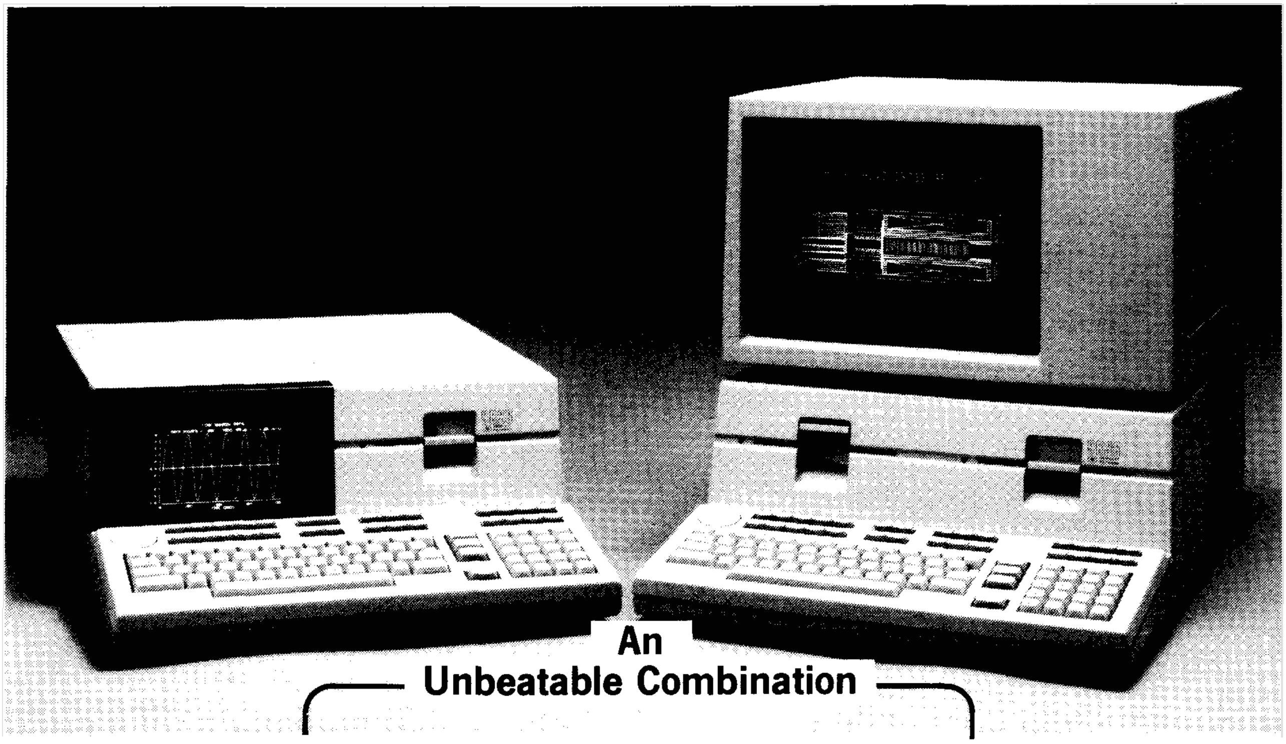
```
Screen # 36          crc ver = 53315
( snapshot debugger                                     10/02/82 )
: SEE    3 SPACES  DUP NFA ID.  ." ="  @ 5 .R ;
VARIABLE 'SNAP
: (SNAP)    6 0 XY  XCUR SEE  YCUR SEE   LENGTH SEE
   RIGHT SEE  TALLY SEE   3 SPACES  ." R# "  R# ?
      YCUR @  #LINES 0 DO   I YCUR !   71 I 2+     XY
   EDGE @  4 .R   EDGE 2+ @  5 .R  LOOP   YCUR !  AT ;
( use [']  instead of ' in Starting FORTH systems: )
: W/SNAP    ' (SNAP)  'SNAP ! ;
: WO/SNAP    ' TASK   'SNAP ! ;

( include CFA in fig-FORTH systems:)
: SNAP   'SNAP @ ( CFA) EXECUTE ;

WO/SNAP
```

```
Screen # 37          crc ver = 31510
   ( Enter mode  BURST I/O VERSION                      07/30/82 )
CREATE HOLDING  1026 ALLOT  ( insert buffer; cell 0 = count)
: ENTER   ( go into Enter mode)
   R# @ 1024 < IF ENTERING MODE !  CLR->SCR  BOTTOM ." Insert:>"
   CHARACTER DUP  ROOM 60 MIN TYPE  HOLDING 2+ ROOM CMOVE
   ROOM HOLDING !  .MODE  AT  0 TALLY !  THEN ;
: ICHAR ( insert one char.)   R# @  1024 < IF  #KEY @ DUP
   CHARACTER C!  ECHO  -1 CHANGE  THEN ;
: eSTOP ( end Enter mode)
   MOOT SHIFTS !  HOLDING REATTACH ;
: e< ( backspace in Enter mode)
   R# @  IF BACKWARD  AT SPACE  1 CHANGE  THEN ;
30 LOAD   ( continue loading remainder of application)
```

10/13/82

*Now Available On*

## HEWLETT PACKARD DESKTOP COMPUTERS



An Unbeatable Combination

### HARDWARE

The HP 9826A and 9836A are two of Hewlett-Packard's newest and most powerful desktop computers. Each is based on the Motorola MC68000 microprocessor. Both machines have full graphics capability and up to 2 full megabytes of user read/write memory. Both operate on 5¼" flexible disc drives (the 9836A has two) which feature 264K bytes of mass storage. While the 9826A has an integral 7" (178mm) CRT which makes it useful for computer-aided testing (CAT) and control, the 9836A has a full 12.2" (310mm) CRT which makes it ideal for computer-aided engineering (CAE) applications. Each model features the following:

- Seven levels of prioritized interrupt
- Memory-mapped I/O
- Built-in HP-IB interface
- Standard ASCII keyboard with numeric keypad and international language options
- Ten (20 with shift) user-definable soft keys with soft labels
- Rotary-control knob for cursor control, interrupt generation and analog simulations
- System clock and three timers
- Powerfail recovery option for protection against power lapses
- Seven additional interface cards
  - — DMA controller (up to 2.4 mb/sec)
  - — 8/16 bit bi-directional parallel
  - — Additional HPIB interface
  - — Serial RS232/449
  - — BCD
  - — Color video(RGB) 3 planes 512 x 512 8 color

### SOFTWARE

**HP 9826/36 Multi-FORTH**
**HP PRODUCT # 97030JA**

Multi-FORTH was developed in 1979 by Creative Solutions, Inc. The standard product has been substantially modified to take full advantage of the 9826/36 hardware features.

**Multi-FORTH features**
- 79 standard programming environment
- Multitasking
- Full screen editor
- In-line structured assembler
- I/O and graphics extensions
- Loadable H.P. floating point (IEEE format)
- Extensive user manuals and documentation

**Optional Features:**
- Meta compiler
- Multi user
- Data access methods library

This product is part of HP PLUS — a program for locating user software. It has been developed by an independent software supplier to run on HP computer systems. It is eligible for HP PLUS as determined by references from satisfied end users. Support services are available only through the software supplier. Hewlett-Packard's responsibilities are described in the Responsibilities Statement below.

**Responsibilities Statement**
*HP PLUS software was developed by an independent software supplier for operation on HP computer systems. The supplier is solely responsible for its software and support services. HP is not the manufacturer or developer of such software or support. HP disclaims any and all liabilities for and makes no warranties, expressed or implied, with respect to this software. Distribution of this product or information concerning this product does not constitute endorsement of the product, the supplier, or support services.* **The customer is responsible for selection of the software it purchases.**

For more information, please write

**Marvel Ross, Hewlett Packard Company**

3404 East Harmony Road, Ft. Collins, CO. 80525

# SSI-FORTH

SSI*FORTH for the IBM Personal Computer is fig-FORTH and much more. Our FORTH system is designed for the serious PC programmer and offers powerful editing and debugging capabilities and a complete interface to BIOS and DOS for the keyboard, screen, disks, communication ports, etc. In addition, SSI*FORTH can accept text from other text editors (like our text editor P-Edit), and perform screen to text converions. All screens used are DOS format compatible.

## Editor

- Full Screen FORTH Editor with Search
- Machine Speed Search Over Multiple Screens

## Additions to FIG FORTH

- Complete Interface to BIOS and DOS for Keyboard, Screen, Disks, Communication Ports, etc.
- The words PEEK, POKE, PEEKW, POKEW for easy transfer of functions defined in the BASIC language.
- Ability to use the speaker and timer to produce tones.

## Debugging Capabilities

- Debugging Aid
- HEX Dump
- Decompiler

## Text Conversion

- Screen to Text Document Conversion
- Load from Text Format

## Requirements

- IBM Personal Computer
- PC/DOS

Price $95

SSI

SATELLITE SOFTWARE INTERNATIONAL
288 WEST CENTER OREM, UTAH 84057

(801) 224-8554 (800) 321-5906 TELEX 453-168

# Representation for Logical True

*Robert L. Smith*

One of the proposals for the next FORTH Standard is that the system default value for "true" is all 16 bits set to "1." The system default for "false" would be unchanged, namely the value returned on the stack has the 16 bits set to "0." FORTH-79 returns a default "true" value with only the low order bit set to "1" and the remaining bits set to "0." Since logical operators in FORTH usually manipulate 16 bit quantities rather than 1 bit quantities, it makes somewhat better sense for the default truth values to return either all bits cleared or all bits set. It should be noted that in many cases the implementation of all ones as a default true is somewhat more efficient than 15 "0"s and a "1." Frequently there is also some improvement in the high level FORTH code.

For a trivial example, consider the function discussed in a previous issue, **S->D**. This is used to "sign-extend" a single precision signed number to a double precision format. Using the suggested logical default, the function may be written:

```
: S->D   DUP 0< ;
```

Using the older default for true requires an additional term, **NEGATE**, to be placed after the **0<**.

An often used trick in FORTH is for a function to either return a zero value or a value, such as an address, based on a logical result. If the new default is used, the logical **AND** operator may be applied to the specified value to give the desired result. To apply the same idea to the older default, we must either perform an arithmetic multiplication, or first negate the logical result before applying the **AND** function.

Probably the main argument against the proposal is that it is not "compatible" with some currently written code. It will be interesting to see the results of the voting on this issue. □

*Editor's Note:*
*Just before this issue went to press, the Standards Team voted to adopt this proposal.*

**1 proFORTH COMPILER**
**8080/8085, Z80 VERSIONS**
- SUPPORTS DEVELOPMENT FOR DEDICATED APPLICATIONS
- INTERACTIVELY TEST HEADERLESS CODE
- IN-PLACE COMPILATION OF ROMABLE TARGET CODE
- MULTIPLE, PURGABLE DICTIONARIES
- FORTH-79 SUPERSET
- AVAILABLE NOW FOR TEKTRONIX DEVELOPMENT SYSTEMS — $2250

**2 MICROPROCESSOR-BASED PRODUCT DESIGN**
- SOFTWARE ENGINEERING
- DESIGN STUDIES — COST ANALYSIS
- ELECTRONICS AND PRINTED CIRCUIT DESIGN
- PROTOTYPE FABRICATION AND TEST
- REAL-TIME ASSEMBLY LANGUAGE/proFORTH
- MULTITASKING
- DIVERSIFIED STAFF

**MICROSYSTEMS, INC.**
**(213) 577-1471**
**2500 E. FOOTHILL BLVD., SUITE 102, PASADENA, CALIFORNIA 91107**

# FORTH-32™
# IBM® PC

Now you can program using the entire IBM® PC memory. The FORTH-32 SEGMENT SENSING language makes segment boundaries transparent to the programmer.

The FORTH-32™ DEVELOPMENT SYSTEM features intermixed 16 and 32 bit addressing modes with fig-FORTH compatibility, DOS interface, full screen editor, assembler, disassembler, graphics and debug. The powerful QUEST CASE statement with single, multiple and range labeling is also included.

In addition to FORTH-32™ you will receive the QUEST PACKAGE BUILDER UTILITY which transforms user developed programs into marketable software packages by building on disk a condensed executable image with only those FORTH words needed.

User oriented self-teaching documentation! All software is fully backed and updated for one year.

$100 FORTH-32 (SPECIFY PC-DOS, OR CP/M-86™)
$50 FLOATING POINT LIBRARY (SOFTWARE OR 8087)

FORTH-32 AND QUEST ARE TRADEMARKS OF QUEST RESEARCH INC.
IBM IS A REGISTERED TRADEMARK OF IBM CORPORATION.
CP/M-86 IS A TRADEMARK OF DIGITAL RESEARCH INC.

Quest™

master charge
VISA
C.O.D.

**QUEST RESEARCH, INC.**
**P.O. BOX 2553**
**HUNTSVILLE, AL 35804**

Call Toll Free:
(800) 558-8088
In Alabama Call:
(205) 533-9405

# Choosing Names

*Henry Laxen*

This time I would like to rant and rave about one of the most difficult aspects of programming in FORTH, that of choosing good names for your definitions. Besides a rational design, this is the single most important part of programming in FORTH.

That's a strong statement, but it is absolutely true. The names you give your definitions can make the difference between understandable, modifiable code, and complete garbage. I will illustrate this by some examples and some guidelines of how to choose good names.

First a word on programming tools. There has been a great deal of time and effort devoted to the topic of programming tools in recent years, and FORTH is well equipped with some of the most sophisticated tools in the software world. You can find code for countless debuggers, decompilers, cross reference utilities, glossary generators, and online helpers of one form or another. These are all wonderful, but rarely is the most important FORTH development tool mentioned, yet it is widely available and costs only about $20.00. I am of course talking about a good dictionary and thesaurus. When it comes to choosing a good name for a FORTH word, these can be invaluable, and should be part of every FORTH programmer's tool kit.

Now then, rule number one in choosing good names is: *Name the what, not the how.* Let's take a look at some examples of what this means. Every FORTH programmer, myself included, is guilty of violating this rule, and the primary violation is in the area of returning booleans or truth values. Every piece of code I have ever seen has phrases such as the following:

```
IF DO-SOMETHING 1 ELSE DROP 0 THEN
```

This is horrible! Furthermore there is a proposal in the 83 Standard to change the value of a true boolean from 1 to −1. If that happens, many many programs will need to be heavily modified. What we have in essence done in the above example is violate our rule on naming clarity. We have named the words TRUE and FALSE with the how, namely **1** and **0**, instead of the what, namely **TRUE** and **FALSE**. A much better solution, and it is absolutely trivial to implement, is to revise the code as follows:

```
1 CONSTANT TRUE
0 CONSTANT FALSE
IF  DO-SOMETHING TRUE ELSE
  DROP FALSE THEN
```

This is great! First it is absolutely clear that we are returning a boolean value, and secondly if this was done throughout, changing the value of **TRUE** would be little more than redefining the constant **TRUE**. The result is clearer, more understandable, and more modifiable code than before. [Editor's note: Or consider using **T** and **F** as abbreviations. See the code for my QTF article on page 21 of this issue.]

Let's look at another example of naming the what and not the how. It is often desirable to define some words which will set a variable to 1 or 0, **TRUE** or **FALSE**. Which of the following pieces of code have you written, and which do you now think is better:

```
: 0! ( addr -- ) 0 SWAP ! ;
: 1! ( addr -- ) 1 SWAP ! ;

: SET ( addr -- ) TRUE SWAP ! ;
: RESET ( addr -- ) FALSE SWAP ! ;
```

Suppose we had a variable called **ENABLE**. Which of the following phrases do you think makes more sense:

**ENABLE 0!** or **ENABLE RESET**

If you ask yourself *how* am I going to disable something, you will come up with the **0!** name. If you ask yourself *what* am I going to do , the answer will be to **RESET** the **ENABLE** flag, and you will come up with the much superior name of **RESET** instead of **0!**.

Always remember to ask yourself what you are doing, not how you are doing it. If you answer the what question, you will most likely come up with a good name.

Now let's proceed to rule number two in how to choose a good name. Rule 2 is: *If possible, stick to English.* Given the choice between good ordinary, prosaic English and super sophisticated computerese, always choose English.

***Besides a rational design, choosing good names is the most important part of programming in FORTH.***

Let's take a look at an example of this rule. What name would you give to the word that takes a row and column position off the stack and moves the cursor of your terminal to that position? Think about it for a minute before you read the next paragraph.

If you chose a word like **GOTOXY** or **XYPOS** may you burn in the fires of PASCAL forever! These are total computerese gibberish, and should be avoided like the plague. A terrific word for this function would be **AT**, since you are positioning the cursor **AT** the values that are on the stack. (This name was stolen by me from Kim Harris who credits Chuck Moore.) Compare how much more nicely the code fragment:

**5 20 AT ." Hello"** reads compared to
**5 20 GOTOXY ." Hello"**

Let's take another example, which might be sacrosanct to many of you. Suppose you wanted to define a word which will list all of the words in a particular vocabulary on your terminal. What would be a good name for such a beast? If you said **VLIST** try again. **VLIST** is another example of computerese gibberish. If you would like to know what the **EDITOR WORDS** are doesn't it make more sense to type **EDITOR WORDS** than **EDITOR VLIST**? **WORDS** is the perfect name for such a function. It names the what, namely

*Continued on page 35*

## JOIN THE APPLICATION MIGRATION!

- PRODUCE MACHINE TRANSPORTABLE CODE.
- GENERATE ROMABLE/HEADERLESS CODE.
- FORWARD REFERENCING ALLOWED.
- PUT FORTH ON OTHER COMPUTERS.
- PRODUCE EXECUTABLE IMAGE IN RAM OR ON DISK.
- PRODUCE ADDRESS MAP OF APPLICATION.
- NO LICENSE FEE OR ROYALTIES ON APPLICATIONS.

fig-FORTH CROSS-COMPILERS by NAUTILUS SYSTEMS
Apple, Atari, TRS-80 Model I, Zenith, and Northstar
fig-FORTH CROSS-COMPILER by LABORATORY MICROSYSTEMS
CP/M-80, CP/M-86, IBM P.C., and 68000 (requires LAB FORTH at additional cost)
79-Standard Systems by MOUNTAIN VIEW PRESS
CP/M-80
$300.00 + tax and $5.00 shipping and handling

Apple is a trademark of Apple Computer, Inc. Atari is a trademark of Atari Computer. TRS-80 is a trademark of Tandy, Corp. Zenith is a trademark of Zenith Radio Corporation. Northstar is a trademark of Northstar Computers. IBM is a trademark of International Business Machines, Inc.

**Nautilus Systems**
P.O. Box 1098 SANTA CRUZ, CA 95061

## CCI MegaFORTH for Data General Computers

* *FORTH-79 superset and various utilities*
* *Multitasking, Multiuser*
* *Cross compiler - can produce ROMable code*
* *Cross reference generator*
* *Screen and line editors*
* *Floating point support*
* *Works on NOVA and ECLIPSE computers including microNOVA and microECLIPSE*
* *Uses ECLIPSE instruction set on ECLIPSEs*
* *Runs under DG operating systems*
* *Extensive user manual and documentation*

The CCI version of fig-FORTH for the **NOVA** is available at **$90.00**

**CAPSTONE COMPUTING, INC.**
5640 Southwyck Blvd., Suite 2E
Toledo, Ohio 43614
Tel. (419) 866-5503

tell me what the **WORDS** are, not the how of Vocabulary LISTing.

Now let's take a look at the third and final rule in choosing a good name. Rule number 3 is: *All things being equal between two names, choose the shorter one.* Let's try our rules on the following problems. Think of a name for a word that will clear the screen of a video terminal. Some names that immediately spring to mind are: **ERASE**, **BLANK** and **CLEAR**. Unfortunately **ERASE** and **BLANK** are already taken, and **CLEAR** seems like a good choice, but maybe we can do better. **CLEAR** could apply to other things besides a video terminal, so think about words that would only apply to visual things. Consider the word **DARK**. This is ideal for this function. All things being equal between **CLEAR** and **DARK** we would choose **DARK** based on rule 3. Let's look at one more example. What name should I give the word that decompiles other FORTH words. The syntax I want is

??? **NAME**

where ??? will decompile the FORTH word **NAME**. Think of what we are doing and come up with some names. Rule 2 excludes garbage such as **DECOMP** and **DIS**. What is it we are doing? We are exposing the definition of **NAME**. Think of words that mean expose. How about the following: **EXPOSE DISCLOSE REVEAL SEE**. They are all good English words that describe what is going on.

For a long time I used **REVEAL** for this function, but then later I finally came up with **SEE**, and chose it based on rule number 3. I don't see any intrinsic value of **SEE** over **REVEAL** other than it is shorter, and hence easier to type. Both **SEE QUIT** and **REVEAL QUIT** appeal to me.

As a final example, and perhaps a piece of useful code that you can use in your applications, let's take a look at Fig. 1. This example was motivated by a frequent occurrence in many of my programs, namely that of returning a TRUE or FALSE result during some kind of searching procedure. Furthermore, this returned result must be capable of nesting properly.

For example, suppose we wanted to search a string for an occurrence of a control character. If you are passed the address and length of the string, you might wind up with a piece of code as shown in Fig. 2.

It first shoves a **0** behind the address and length on the stack. It then runs through the string character by character and if it finds a control character, it throws away the current address and the **0**, replaces them with two **1**s, and leaves the loop. After the loop, the address is thrown away, leaving only the boolean result. I think this is not only hard to follow, but tricky, and should be avoided.

Now compare it with the piece of code in Fig. 3. It starts out by saying that the result to be returned is initially false. Next it also runs through the string character by character, and if it finds a control character it simply indicates that the search was successful and leaves. After the loop the address is thrown away and the result is returned. What could be simpler and more readable?

Now let's examine Fig. 1 in more detail. The word **INITIALLY** is nothing more than a push onto a stack pointed to by the word **BOOLEANS**. Similarly **RESULT** is nothing more than a pop from the same stack. Notice how completely different the names are from how they are actually implemented. If I had named the how instead of the what, I would have wound up with names like **>BOOL** and **BOOL>**. Not only would this violate rule number 1, but it would be complete computer gibberish as well. How many of you have implemented stacks with names such as **>GARBAGE** and **GARBAGE>**?

Just because something is a stack doesn't mean it has to have little arrows associated with it. Stacks are very useful data structures, and when you use them to implement a function, be sure to name them according to what the function does, not how it does it. I have found that using the code in Fig. 1 has improved the readability of my programming immensely, at almost zero cost.

In conclusion, I would like to leave you with the immortal words of the poet John Keats, from his poem ENDYMION. He said something like: "A good name is a joy forever." Till next time, may the FORTH be with you. □

*Henry Laxen is an independent FORTH consultant based in Berkeley, California.*

```
Scr # 1
\ Fig 1.       Boolean Results                             11SEP82HHL
CREATE BOOLEANS   0 ,    20 ALLOT    ( Space for the stack )
: INITIALLY    ( n -- )
   BOOLEANS   2 OVER +!    ( increment index )
   DUP @ + !    ( and store n ).   ;
: RESULT       ( n -- )
   BOOLEANS   DUP   DUP @ + @ ( get top of stack )
   -2 ROT +! ( and decrement index )   ;
: FAIL         ( -- )
   RESULT DROP   FALSE INITIALLY   ;
: SUCCEED      ( -- )
   RESULT DROP    TRUE INITIALLY   ;
```

```
Scr # 2
\ Fig. 2.   Poor way to Search a String                   11SEP82HHL
: CONTROL?   ( addr len -- f )
   0 ROT ROT    0 DO    DUP C@ BL
      < IF    2DROP    1 1    LEAVE    THEN
   LOOP    DROP   ;


\ Fig. 3.   Neat way to Search a String
: CONTROL?   ( addr len -- f )
   FALSE INITIALLY    0 DO    DUP C@ BL
      < IF    SUCCEED    LEAVE    THEN
   LOOP    DROP    RESULT   ;
```

## TAKE FORTH TO WORK

NOW YOU CAN RUN FORTH ON THE OFFICE IBM MAINFRAME. GIVE YOURSELF THE FORTH ADVANTAGE ON THE JOB!

## FORTH/370

**for large IBM and equivalent computers**

- IBM 370, 4341, 3033, etc.
- Based on fig FORTH
- Program compatible with micro FORTH systems
- Editor and Assembler
- Runs under VM/CMS or MVS/TSO
- 32 bit word, 64 bit double word
- Files compatible with host operating system

ASK YOUR EMPLOYER TO BUY FORTH/370 AND START USING FORTH 8 HOURS PER DAY.

Introductory Price $1350.
5% Sales Tax in Maryland

**WARD SYSTEMS GROUP**
8013 Meadowview Drive
Frederick, Maryland 21701
(301) 695-8750

## 1983 ROCHESTER FORTH APPLICATIONS CONFERENCE With a Focus on Robotics

June 7 through June 11, 1983

University of Rochester
Rochester, New York

- The third annual Rochester Forth Conference will be hosted by the University of Rochester's Laboratory for Laser Energetics and sponsored by the Institute for Applied Forth Research, Inc. This year's conference has a format similar to that of previous Rochester conferences with an emphasis on Forth applications focused on a special topic. This year's topic is robotics, which embraces many areas including, but not limited to: mechanical and electrical engineering, vision, artificial intelligence, computer networking and automated manufacturing. We believe that the nature of Forth and its application to robotics provides a unique opportunity to study both disciplines.
- There is a call for papers on the following topics:
  1. Robotics and Forth.
  2. Forth applications, including, but not limited to: real time, business, medical, space-based, laboratory and personal systems; and Forth microchip applications.
  3. Forth technology, including finite state machines, control structures, and defining words.
- Papers will be handled in either oral or poster sessions, although oral papers will be refereed in accordance with conference direction and paper suitability. Please submit a 200 word abstract by April 15, 1983. Papers for the oral session must be received by May 15 and for poster sessions by June 1, 1983. Papers are limited to a maximum of 10 printed pages including code and figures. If this restriction causes a problem, please contact us.
- For more information, please contact the conference chairman:

  Lawrence P. Forsley
  Laboratory for Laser Energetics
  250 East River Road
  Rochester, New York 14623

# Fig Chapters

## U.S.

### • ARIZONA

**Phoenix Chapter**
Peter Bates at 602/996-8398

### • CALIFORNIA

**Los Angeles Chapter**
Monthly, 4th Sat., 11 a.m., Allstate Savings, 8800 So. Sepulveda Blvd., L.A. Philip Wasson 213/649-1428

**Northern California Chapter**
Monthly, 4th Sat., 1 p.m., FORML Workshop at 10 a.m. Palo Alto area. Contact FIG Hotline 415/962-8653

**Orange County Chapter**
Monthly, 3rd Sat., 12 noon, Fullerton Savings, 18020 Brockhorst, Fountain Valley. 714/896-2016

**San Diego Chapter**
Weekly, Thurs., 12 noon. Call Guy Kelly, 714/268-3100 x4784

### • MASSACHUSETTS

**Boston Chapter**
Monthly, 1st Wed., 7 p.m. Mitre Corp. Cafeteria, Bedford, MA. Bob Demrow, 617/688-5661 after 5 p.m.

### • MICHIGAN

**Detroit Chapter**
Call Dean Vieau, 313/493-5105

### • MINNESOTA

**MNFIG Chapter**
Monthly, 1st Mon. Call Mark Abbot (days) 612/854-8776 or Fred Olson, 612/588-9532, or write to: MNFIG, 1156 Lincoln Ave., St. Paul, MN 55105

### • NEW JERSEY

**New Jersey Chapter**
Call George Lyons, 201/451-2905 eves.

### • NEW YORK

**New York Chapter**
Call Tom Jung, 212/746-4062

### • OKLAHOMA

**Tulsa Chapter**
Monthly, 3rd Tues., 7:30 p.m., The Computer Store, 4343 So. Peoria, Tulsa, OK. Call Bob Giles, 918/599-9304 or Art Gorski, 918/743-0113

### • OHIO

**Dayton Chapter**
Monthly, 2nd Tues., Datalink Computer Center, 4920 Airway Road, Dayton, OH 45431. Call Gary Ganger, (513) 849-1483.

### • OREGON

**Portland Chapter**
Call Timothy Huang, 9529 Northeast Gertz Circle, Portland, OR 97211, 503/289-9135

### • PENNSYLVANIA

**Philadelphia Chapter**
Call Barry Greebel, Continental Data Systems, 1 Bala Plaza, Suite 212, Bala Cynwid, PA 19004

### • TEXAS

**Austin Chapter**
Call John Hastings, 512/327-5864

**Dallas/Ft. Worth Chapter**
Monthly, 4th Thurs. 7 p.m., Software Automation, 1005 Business Parkway, Richardson, TX. Call Marvin Elder, 214/231-9142 or Bill Drissel, 214/264-9680

### • UTAH

**Salt Lake City Chapter**
Call Bill Haygood, 801/942-8000

### • VERMONT

**ACE Fig Chapter**
Monthly, 4th Thur., 7:30 p.m., The Isley Library, 3rd Floor Meeting Rm., Main St., Middlebury, VT 05753. Contact Hal Clark, RD #1 Box 810, Starksboro, VT 05487, 802/877-2911 days; 802/453-4442 eves.

### • VIRGINIA

**Potomac Chapter**
Monthly, 1st Tues. 7p.m., Lee Center, Lee Highway at Lexington Street, Arlington, Virginia. Call Joel Shprentz, 703/437-9218 eves.

### • WASHINGTON

**Seattle Chapter**
Call Chuck Pliske or Dwight Vandenburg, 206/542-7611

**Nevada**

**Las Vegas Chapter**
Call Gerald Hasty, 702/737-5670

## FOREIGN

### • AUSTRALIA

**Australia Chapter**
Contact Lance Collins, 65 Martin Rd., Glen Iris, Victoria 3146, or phone (03) 292600

### • CANADA

**Southern Ontario Chapter**
Contact Dr. N. Solnseff, Unit for Computer Science, McMaster University, Hamilton, Ontario L8S 4K1, 416/525-9140 x2065

**Quebec Chapter**
Call Gilles Paillard, 418/871-1960 or 643-2561

### • ENGLAND

**English Chapter**
Write to FORTH Interest Group, 38 Worsley Rd., Frimley, Camberley, Surrey, GU16 5AU, England

### • JAPAN

**Japanese Chapter**
Contact Masa Tasaki, Baba-Bldg. 8F, 3-23-8 Nishi-Shimbashi, Minato-ku, Tokyo, 105 Japan

### • NETHERLANDS

**HCC-FORTH Interest Group Chapter**
Contact F.J. Meijer, Digicos, Aart V.D. Neerweg 31, Ouderkerk A.D. Amstel, The Netherlands

### • WEST GERMANY

**West German Chapter**
Contact Wolf Gervert, Roter Hahn 29, D-2 Hamburg 72, West Germany, (040) 644-3985

## SPECIAL GROUPS

**Apple Corps FORTH Users Chapter**
Twice monthly, 1st & 3rd Tues., 7:30 p.m., 1515 Sloat Blvd., #2, San Francisco, CA. Call Robert Dudley Ackerman, 415/626-6295

**Nova Group Chapter**
Contact Mr. Francis Saint, 2218 Lulu, Witchita, KS 67211, 316/261-6280 (days)

**MMSFORTH Users Chapter**
Monthly, 3rd Wed., 7 p.m., Cochituate, MA. Dick Miller, 617/653-6136

## List of FORTH System Vendors

**(e.g., A1 signifies AB Computers, etc.)**

### Processors

| | |
|---|---|
| 1802 | C1, C2, F3, F6, L3 |
| 6502 (AIM, KIM, SYM) | R1, R2, S1 |
| 6800 | F3, F5, K1, L3, M6, T1 |
| 6809 | F3, F5, L3, M6, T1 |
| 68000 | C4, E1 |
| 8080/85 | A5, C1, C2, F4, I5, L1, L3, M3, M6, R1 |
| Z80/89 | A3, A5, C2, F4, I3, K1, L1, M2, M3, M5, N1 |
| Z8000 | I3 |
| 8086/88 | F2, F3, L1, L3, M6 |
| 9900 | E2, L3 |

### Operating Systems

| | |
|---|---|
| CP/M | A3, C2, F3, I3, L3, M1, M2, M6 |

### Computers

| | |
|---|---|
| Alpha Micro | P3, S3 |
| Apple | A4, F4, I2, I4, J1, L4, M2, M6, O2, O3 |
| Atari | M6, P2, Q1 |
| Cromemco | A5, M2, M6 |
| DEC PDP/LSI-11 | C2, F3, K1, L2, S3 |
| Heath-89 | M2, M6 |
| Hewlett-Packard 85 | |
| IBM PC | C2, F3, L1, M5, M6 |
| IBM Other | L3 |
| Micropolis | A2, M2, S2 |
| North Star | I5, M2, P1, S7 |
| Ohio Scientific | A6, B1, C3, O1, S6, T2 |
| Osborne | |
| Pet SWTPC | A1, A6, B1, C3, O1, S6, T2, T5 |
| TRS-80 I, II, III | I5, M5, M6, S4, S5 |
| TRS-80 Color | A3, F5, M4, T1 |

### Other Products/Services

| | |
|---|---|
| Boards, Machine | F3, M3, R2 |
| Consultation | C2, C4, N1 |
| Cross Compilers | C2, F3, I3, M6, N1 |
| Products, Various | C2, F3, I5, S8 |
| Training | F3, I3 |

---

**FORTH-79**

**Ver. 2 For your APPLE II/II+**

**The complete professional software system, that meets ALL provisions of the FORTH—79 Standard (adopted Oct. 1980). Compare the many advanced features of FORTH—79 with the FORTH you are now using, or plan to buy!**

| FEATURES | OURS | OTHERS |
|---|---|---|
| 79-Standard system gives source portability. | YES | |
| Professionally written tutorial & user manual | 200 PG. | |
| Screen editor with user-definable controls. | YES | |
| Macro-assembler with local labels. | YES | |
| Virtual memory. | YES | |
| Both 13 & 16-sector format. | YES | |
| Multiple disk drives. | YES | |
| Double-number Standard & String extensions. | YES | |
| Upper/lower case keyboard input. | YES | |
| LO-Res graphics. | YES | |
| 80 column display capability | YES | |
| Z-80 CP/M Ver. 2.x & Northstar also available | YES | |
| Affordable! | $99.95 | |
| Low cost enhancement option: | | |
| Hi-Res turtle-graphics. | YES | |
| Floating-point mathematics. | YES | |
| Powerful package with own manual, 50 functions in all, AM9511 compatible. | | |

| | |
|---|---|
| FORTH—79 V.2 (requires 48K & 1 disk drive) | $ 99.95 |
| ENHANCEMENT PACKAGE FOR V.2 Floating point & Hi-Res turtle-graphics | $ 49.95 |
| COMBINATION PACKAGE | $139.95 |

(CA res. add 6% tax: COD accepted)

**MicroMotion**
12077 Wilshire Blvd. # 506
L.A., CA 90025 (213) 821-4340
Specify APPLE, CP/M or Northstar
Dealer inquiries invited.

---

**FORTH-79**

**Version 2 For Z-80, CP/M (1.4 & 2.x), & NorthStar DOS Users**

**The complete professional software system, that meets ALL provisions of the FORTH—79 Standard (adopted Oct. 1980). Compare the many advanced features of FORTH—79 with the FORTH you are now using, or plan to buy!**

| FEATURES | OURS | OTHERS |
|---|---|---|
| 79-Standard system gives source portability. | YES | |
| Professionally written tutorial & user manual. | 200 PG. | |
| Screen editor with user-definable controls. | YES | |
| Macro-assembler with local labels. | YES | |
| Virtual memory. | YES | |
| BDOS, BIOS & console control functions (CP/M). | YES | |
| FORTH screen files use standard resident file format. | YES | |
| Double-number Standard & String extensions. | YES | |
| Upper/lower case keyboard input. | YES | |
| APPLE II/II+ version also available. | YES | |
| Affordable! | $99.95 | |
| Low cost enhancement options; | | |
| Floating-point mathematics | YES | |
| Tutorial reference manual | | |
| 50 functions (AM9511 compatible format) | | |
| Hi-Res turtle-graphics (NoStar Adv. only) | YES | |

| | |
|---|---|
| FORTH-79 V.2 (requires CP/M Ver. 2.x). | $99.95 |
| ENHANCEMENT PACKAGE FOR V.2: Floating point | $ 49.95 |
| COMBINATION PACKAGE (Base & Floating point) | $139.95 |

(advantage users add $49.95 for Hi-Res)
(CA. res. add 6% tax; COD & dealer inquiries welcome)

**MicroMotion**
12077 Wilshire Blvd. # 506
L.A., CA 90025 (213) 821-4340
Specify APPLE, CP/M or Northstar
Dealer inquiries invited.

# FORTH Vendors

The following vendors offer FORTH systems, applications, or consultation. FIG makes no judgement on any product, and takes no responsibility for the accuracy of this list. We encourage readers to keep us informed on availability of the products and services listed. Vendors may send additions and corrections to the Editor, and must include a copy of sales literature or advertising.

## FORTH Systems

**A**

1. AB Computers
 252 Bethlehem Pike
 Colmar, PA 18915
 215/822-7727
2. Acropolis
 17453 Via Valencia
 San Lorenzo, CA 94580
 415/276-6050
3. Advanced Technology Corp.
 P.O. Box 726
 Clinton, TN 37716
4. Applied Analytics Inc.
 8910 Brookridge Drive, #300
 Upper Marlboro, MD 20870
5. Aristotelian Logicians
 2631 East Pinchot Avenue
 Phoenix, AZ 85016
6. Aurora Software Associates
 P.O. Box 99553
 Cleveland, OH 44199

**B**

1. Blue Sky Products
 729 E. Willow
 Signal Hill, CA 90806

**C**

1. CMOSOFT
 P.O. Box 44037
 Sylmar, CA 91342
2. COMSOL, Ltd.
 Treway House
 Hanworth Lane
 Chertsey, Surrey KT16 9LA
 England
3. Consumer Computers
 8907 La Mesa Boulevard
 La Mesa, CA 92041
 714/698-8088
4. Creative Solutions, Inc.
 4801 Randolph Road
 Rockville, MD 20852

**D**

1. Datentec Kukulies
 Heinrichsallee 35
 Aachen, 5100
 West Germany

**E**

1. Emperical Research Group
 P.O. Box 1176
 Milton, WA 98354
 206/631-4855
2. Engineering Logic
 1252 13th Avenue
 Sacramento, CA 95822

**F**

1. Fantasia Systems, Inc.
 1059 Alameda De Las Pulgas
 Belmont, CA 94002
 415/593-5700
2. Fillmore Systems
 5227 Highland Road
 Minnetonka, MN 55343
3. FORTH, Inc.
 2309 Pacific Coast Highway
 Hermosa Beach, CA 90254
 213/372-8493
4. FORTHWare
 639 Crossridge Terrace
 Orinda, CA 94563
5. Frank Hogg Laboratory, Inc.
 130 Midtown Plaza
 Syracuse, NY 13210
 315/474-7856
6. FSS
 P.O. Box 8403
 Austin, TX 78712
 512/477-2207

**I**

1. IDPC Company
 P.O. Box 11594
 Philadelphia, PA 19116
 215/676-3235
2. IUS (Cap'n Software)
 281 Arlington Avenue
 Berkeley, CA 94704
 415/525-9452
3. Inner Access
 517K Marine View
 Belmont, CA 94002
 415/591-8295
4. Insoft
 10175 S.W. Barbur Blvd., #202B
 Portland, OR 97219
 503/244-4181
5. Interactive Computer
 Systems, Inc.
 6403 Di Marco Road
 Tampa, FL 33614

**J**

1. JPS Microsystems, Inc.
 361 Steelcase Road, West, Unit 1
 Markham, Ontario,
 Canada L3R 3V8
 416/475-2383

**L**

1. Laboratory Microsystems
 4147 Beethoven Street
 Los Angeles, CA 90066
 213/306-7412
2. Laboratory Software
 Systems, Inc.
 3634 Mandeville Canyon Road
 Los Angeles, CA 90049
 213/472-6995
3. Lynx
 3301 Ocean Park, #301
 Santa Monica, CA 90405
 213/450-2466
4. Lyons, George
 280 Henderson Street
 Jersey City, NJ 07302
 201/451-2905

**M**

1. M & B Design
 820 Sweetbay Drive
 Sunnyvale, CA 94086
2. MicroMotion
 12077 Wilshire Boulevard, #506
 Los Angeles, CA 90025
 213/821-4340
3. Microsystems, Inc.
 2500 E. Foothill Boulevard, #102
 Pasadena, CA 91107
 213/577-1417
4. Micro Works, The
 P.O. Box 1110
 Del Mar, CA 92014
 714/942-2400
5. Miller Microcomputer Services
 61 Lake Shore Road
 Natick, MA 01760
 617/653-6136
6. Mountain View Press
 P.O. Box 4656
 Mountain View, CA 94040
 415/961-4103

**N**

1. Nautilus Systems
 P.O. Box 1098
 Santa Cruz, CA 95061
 408/475-7461

**O**

1. OSI Software & Hardware
 3336 Avondale Court
 Windsor, Ontario
 Canada N9E 1X6
 519/969-2500
2. Offete Enterprises
 1306 S "B" Street
 San Mateo, CA 94402
3. On-Going Ideas
 RD #1, Box 810
 Starksboro, VT 05487
 802/453-4442

**P**

1. Perkel Software Systems
 1636 N. Sherman
 Springfield, MO 65803
2. Pink Noise Studios
 P.O. Box 785
 Crockett, CA 94525
 415/787-1534
3. Professional Management
 Services
 724 Arastradero Road, #109
 Palo Alto, CA 94306
 408/252-2218

**Q**

1. Quality Software
 6660 Reseda Boulevard, #105
 Reseda, CA 91335

**R**

1. Rehnke, Eric C.
 540 S. Ranch View Circle, #61
 Anaheim Hills, CA 92087
2. Rockwell International
 Microelectronics Devices
 P.O. Box 3669
 Anaheim, CA 92803
 714/632-2862

**S**

1. Saturn Software, Ltd.
 P.O. Box 397
 New Westminister, BC
 V3L 4Y7 Canada
2. Shaw Labs, Ltd.
 P.O. Box 3471
 Hayward, CA 94540
 415/276-6050
3. Sierra Computer Co.
 617 Mark NE
 Albuquerque, NM 87123
4. Sirius Systems
 7528 Oak Ridge Highway
 Knoxville, TN 37921
 615/693-6583
5. Software Farm, The
 P.O. Box 2304
 Reston, VA 22090
6. Software Federation
 44 University Drive
 Arlington Heights, IL 60004
 312/259-1355
7. Software Works, The
 1032 Elwell Court, #210
 Palo Alto, CA 94303
 415/960-1800
8. Supersoft Associates
 P.O. Box 1628
 Champaign, IL 61820
 217/359-2112

**T**

1. Talbot Microsystems
 1927 Curtis Avenue
 Redondo Beach, CA 90278
2. Technical Products Co.
 P.O. Box 12983
 Gainsville, FL 32604
 904/372-8439
3. Timin Engineering Co.
 6044 Erlanger Street
 San Diego, CA 92122
 714/455-9008
4. Transportable Software, Inc.
 P.O. Box 1049
 Hightstown, NJ 08520
 609/448-4175

**Z**

1. Zimmer, Tom
 292 Falcato Drive
 Milpitas, CA 95035

## Boards & Machines Only

see System Vendor Chart for others

Controlex Corp.
16005 Sherman Way
Van Nuys, CA 91406
213/780-8877

Datricon
7911 NE 33rd Drive, #200
Portland, OR 97211
503/284-8277

Golden River Corp.
7315 Reddfield Court
Falls Church, CA 22043

Peopleware Systems Inc.
5190 West 76th Street
Minneapolis, MN 55435
612/831-0872

Zendex Corp.
6398 Dougherty Road
Dublin, CA 94566

## Application Packages Only

see System Vendor Chart for others

R. E. Curry & Associates
P.O. Box 11324
Palo Alto, CA 94306

InnoSys
2150 Shattuck Avenue
Berkeley, CA 94704
415/843-8114

## Consultation & Training Only

see System Vendor Chart for others

Boulton, Dave
581 Oakridge Drive
Redwood City, CA 94062

Brodie, Leo
9720 Baden Avenue
Chatsworth, CA 91311
213/998-8302

Girton, George
1753 Franklin
Santa Monica, CA 90404
213/829-1074

Go FORTH
504 Lakemead Way
Redwood City, CA 94062
415/366-6124

Harris, Kim R.
Forthright Enterprises
P.O. Box 50911
Palo Alto, CA 94303
415/858-0933

Laxen, Henry H.
1259 Cornell Avenue
Berkeley, CA 94706
415/525-8582

Petri, Martin B.
15508 Lull Street
Van Nuys, CA 91406
213/908-0160

Redding Co.
P.O. Box 498
Georgetown, CT 06829
203/938-9381

Schleisiek, Klaus
c/o J. Buettuer
Eppeudorfer Landstr. 16
D 2 Hamburg 20
W. Germany

Schrenk, Dr. Walter
Postfach 904
7500 Krlsruhe-41
W. Germany

Software Engineering
317 W. 39th Terrace
Kansas City, MO 64111
816/531-5950

Technology Management, Inc.
1520 S. Lyon
Santa Ana, CA 92705

## FORTH INTEREST GROUP MAIL ORDER

| | USA | FOREIGN AIR |
|---|---|---|
| ☐ Membership in FORTH INTEREST GROUP and Volume IV of FORTH DIMENSIONS (6 issues) | $15 | $27 |
| ☐ Volume III of FORTH DIMENSIONS (6 issues) | 15 | 18 |
| ☐ Volume II of FORTH DIMENSIONS (6 issues) | 15 | 18 |
| ☐ Volume I of FORTH DIMENSIONS (6 issues) | 15 | 18 |
| ☐ fig-FORTH Installation Manual, containing the language model of fig-FORTH, a complete glossary, memory map and installation instructions | 15 | 18 |
| ☐ Assembly Language Source Listing of fig-FORTH for specific CPU's and machines. The above manual is required for installation. Check appropriate boxes. **Price per each.** ☐ 1802 ☐ 6502 ☐ 6800 ☐ 6809 ☐ 8080 ☐ 8086/8088 ☐ 9900 ☐ APPLE II ☐ PACE ☐ NOVA ☐ PDP-11 ☐ ALPHA MICRO | 15 | 18 |
| ☐ "Starting FORTH" by Brodie. BEST book on FORTH. (Paperback) | 16 | 20 |
| ☐ "Starting FORTH" by Brodie. (Hard Cover) | 20 | 25 |
| ☐ PROCEEDINGS 1980 FORML (FORTH Modification Lab) Conference | 25 | 35 |
| ☐ PROCEEDINGS 1981 FORTH University of Rochester Conference | 25 | 35 |
| ☐ PROCEEDINGS 1981 FORML Conference, Both Volumes | 40 | 55 |
| ☐ Volume I, Language Structure | 25 | 35 |
| ☐ Volume II, Systems and Applications | 25 | 35 |
| ☐ FORTH-79 Standard, a publication of the FORTH Standards Team | 15 | 18 |
| ☐ Kitt Peak Primer, by Stevens. An indepth self-study primer | 25 | 35 |
| ☐ BYTE Magazine Reprints of FORTH articles, 8/80 to 4/81 | 5 | 10 |
| ☐ FIG T-shirts: ☐ Small ☐ Medium ☐ Large ☐ X-Large | 10 | 12 |
| ☐ Poster, Aug. 1980 BYTE cover, 16 x 22" | 3 | 5 |
| ☐ FORTH Programmer Reference Card. If ordered separately, send a stamped, addressed envelope. | FREE | |
| TOTAL | $______ | |

NAME____________________ MAIL STOP/APT__________
ORGANIZATION____________________(if company address)
ADDRESS____________________
CITY__________ STATE____ ZIP____ COUNTRY________
VISA #____________ MASTERCARD #____________
EXPIRATION DATE____________ (Minimum of $10.00 on charge cards)

Make check or money order in US Funds on US bank, payable to: **FIG.** All prices include postage. **No purchase orders without check.** California residents add sales tax.

**ORDER PHONE NUMBER: (415) 962-8653**

FORTH INTEREST GROUP PO BOX 1105 SAN CARLOS, CA 94070

**FORTH INTEREST GROUP**
P.O. Box 1105
San Carlos, CA 94070

BULK RATE
U.S. POSTAGE
PAID
Permit No. 261
Mt. View, CA

```
    3582 S
R L SMITH
ESL SUBSIDIAR OF TRW
PO BOX 510
SUNNYVALE, CA 94086
```

**Address Correction Requested**