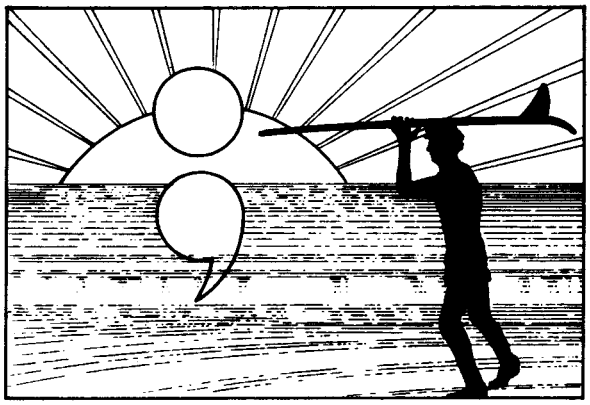
# FORTH Dimensions

Volume V
Number 1
$2.50



**ARTICLES**



**DEPARTMENTS**

# FORTH for Z-80®, 8086, 68000, and IBM® PC

**FORTH Application Development Systems** include interpreter/compiler with virtual memory management and multi-tasking, assembler, full screen editor, decompiler, utilities, and 130+ page manual. Standard random access 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 II ..... $ 50.00
**8080 FORTH** for CP/M 2.2 or MP/M II ..... $ 50.00
**8086 FORTH** for CP/M-86 or MS-DOS ..... $100.00
**PC/FORTH™** for PC-DOS, CP/M-86, or CCPM ..... $100.00
**68000 FORTH** for CP/M-68K ..... $250.00

**FORTH + Systems** are 32 bit implementations that allow creation of programs as large as 1 megabyte. This is the only language that supports the entire memory space of the 8086/88 directly for programs and data!

PC/FORTH + for PC-DOS or CP/M-86 ..... $250.00
8086 FORTH + for CP/M-86 ..... $250.00

**Extension Packages** for FORTH systems

Software floating point (Z-80, 8086, PC only) ..... $100.00
Intel 8087 support (8086, PC only) ..... $100.00
AMD 9511 support (8086, Z-80 only) ..... $100.00
Color graphics (PC only) ..... $100.00
Symbolic interactive debugger (PC only) ..... $100.00
Cross reference utility ..... $ 25.00
PC/GEN™ (custom character sets, PC only) ..... $ 50.00
Hierarchical file manager ..... $ 50.00
B-tree index manager ..... $125.00
B-tree index and file manager ..... $200.00

**QTF +** Screen editor and text formatter by Leo Brodie, for IBM PC with IBM or Epson printer ..... $ 50.00

**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 II), 8086/88 (CP/M-86 or MS-DOS), IBM PC (PC-DOS or CP/M-86), 68000 (CP/M-68K)
Targets: 8080, Z-80, 8086/88, 6502, LSI-11, 68000, 1802, Z-8

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

**AUGUSTA™**, ADA subset compiler from Computer Linguistics, for Z-80 computers under CP/M 2.2 ..... $ 90.00
**LEARNING FORTH** computer-assisted tutorial by Laxen and Harris for CP/M, includes Brodie's "Starting FORTH" ..... $ 95.00
**Z-80 Machine Tests** Memory, disk, printer, and console tests with all source code in standard Zilog mnemonics ..... $50.00
**DATA ACE,** fully relational data base system from CSD, for the IBM Personal Computer. Faster and more powerful than dBASE II ..... $595.00

FORTH application development systems require 48 kbytes RAM and 1 disk drive, Cross-Compilers require 64 kbytes. All software distributed on eight inch, single density, soft sectored diskettes except PC/FORTH on 5¼ inch single sided double density diskettes. Prices include shipping by UPS or first class mail within USA and Canada. California residents add appropriate sales tax. Purchase orders accepted at our discretion.

## Laboratory Microsystems, Inc.
### 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.

Augusta is a trademark of Computer Linguistics
dBASE II is a trademark of Ashton-Tate
PC/FORTH and PC/GEN are trademarks of Laboratory Microsystems Inc.

# From the Editor

This issue of *FORTH Dimensions* marks my last as Editor. Starting with the next issue (V/2), I'm pleased to pass the baton to Marlin Ouverson, formerly the Editor of the distinguished magazine *Dr. Dobb's Journal.* We can all look forward to interesting issues and new ideas from him. Meanwhile, I'll be dividing my time between writing my book on Style and Methodology, and teaching FORTH courses.

I want to thank all of you who contributed your articles and ideas to *FORTH Dimensions* during the past year. Your expertise and dedication has helped make this magazine as useful a journal as it is. In particular I'd like to thank Henry Laxen, a tireless (and I might add, *unpaid*, as are all F.D. writers) columnist, for his always excellent work, and to Robert Smith, FIG's own active archivist on standardization. (Both columnists are taking a much-deserved vacation with this issue, but will return next time.) I'd also like to thank regular reviewers Kim Harris, Michael Perry, Klaxon Suralis, Glen Haydon and Bill Ragsdale for helping me ensure technical correctness of published material.

I'm looking forward to more issues of interesting and pertinent FORTH news and commentary. In fact I just sent off my $15 renewal check. Hope you have, too.

See ya.

*—Leo Brodie*

**Cover Art**

***The end of another beautiful definition as semi-colon sets over the Firth of Forth. —LB***

# Interactive Computer Graphics for Art, Design and Learning

*Bob Gotsch, Time Arts, Inc.*

A "video paint" program called EASEL, written in FORTH for use with several medium-resolution graphics frame buffers (Cromemco and Digital Graphics Systems among others) allows an artist to create images and illustrations with electronic pen on digitizing tablet. Video painting is a new medium of expression with characteristics that challenge an artist's skill and imagination and also provide new creative opportunities.

**Interactive Visual Programming**

Interactive trial and error refinement at computer-displayed images is much more practical than with the sticky paint or dried ink of traditional graphic media. There is no tube of paint that can run out. Moving, scaling, rotation, duplication, and coloring are some of the manipulations available.

Actions are selected in menus that pop up when needed on the color monitor screen. The position of the pen is always shown on the screen as an XORed crosshair; so with menus and positional feedback the artist rarely needs to take attention away from the screen. The hand holding the pen becomes an automatic part of the process of willing an image into being.

Complicated pictures can be built using combinations of line, rectangle, circle, and ellipse primitives, frames and cells loaded from disk, digitized video images, fills and freehand drawing with various size "pens," "airbrushes," and user-defined "brushes." The sequence of actions that will result in the desired image, whether diagrammatic or illusionistic, is often as critical as the sequence of instructions in programming; the sophisticated computer graphic artist is a "visual programmer," using yet a higher level language than FORTH. A tremendous advantage of programming at this level is that any "bugs" are completely visible.

**Extensible Software**

Like FORTH, EASEL is interactive and extensible. The user can select among optional menus and add new menus as they become available, or program special application menus such as TV weather effects, display lettering, or key-frame animation.

Recently added menu operations, "oblique" and "perspective," are used to create depth illusion. The 64K address space isn't nearly adequate for a large and expanding video paint system, so menus (or more accurately the compiled code to perform their

**FORTH Dimensions**

Published by FORTH Interest Group

Volume V, No. 1

May/June 1983

Editorial
Leo Brodie

Publisher
Roy C. Martens

Typesetting/Production
LARC Computing, Inc.



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.

actions) are overlays, loaded quickly when needed. Thus the number of available menu overlays is limited only by disk space.

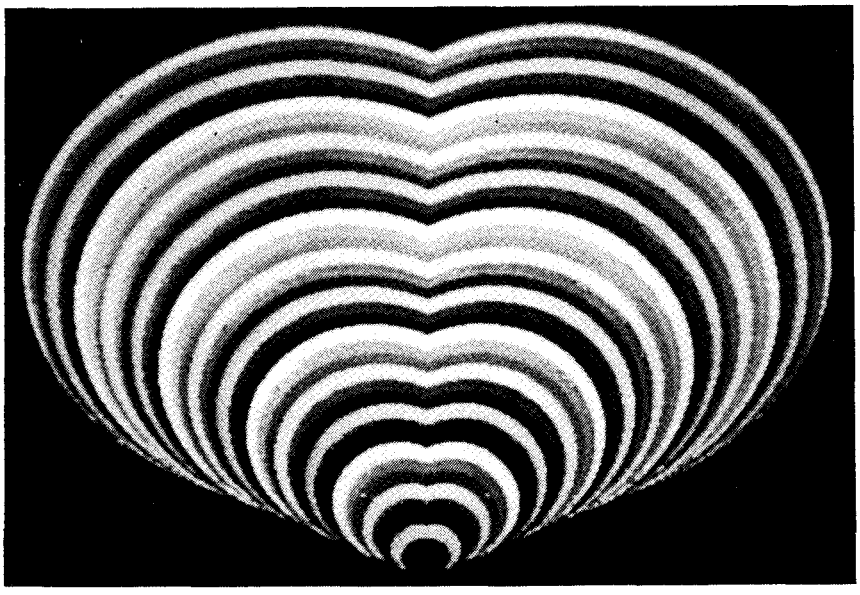
**Access and Applications**

Artists and designers (even the ones who would not be intimidated by computers) have not had much access to high-priced computer graphics equipment. Some applications that currently justify the cost of professional graphics systems are video illustration and real-time animation, story-boarding for film and video, graphic design layout, business graphics, scientific simulation, equipment control and engineering and architectural design.

With cheaper memory and new designs, good frame buffers are becoming available at a fraction of the former cost. This will give many more artists and designers access to interactive graphics systems. And interactive tools are, most of all, useful for learning. Access to tools for experiment rather than production is a necessary part of the creative process. After imitating old graphics for a while with new technology, artists can be expected to contribute new techniques of graphic representation.

This development would have been very difficult using the EXORset's BASIC, not least because the BASIC interpreter uses the graphics display memory, making interactive development of graphics impossible! polyFORTH and its graphics package however fit in about 12K bytes, leaving plenty of room for graphics display, data, and the application program.

# FORTH in the Arts
## Three Application Stories



*Shown here is a print of a digital valentine, a sample of computer art generated by Howard Pearlmutter of Santa Cruz, California. This valentine is also a frame out of a one-minute 16mm movie called "Circlove Life". It was produced on an AED 512 high resolution color system running 6502 FORTH as its operating system, graphics language, and "heart description language." Once the graphics language had been implemented, it took only three days to program the entire animation, including the "heart description language."*

*Howard Pearlmutter spoke at last year's FIG national convention on this project:*

### Digital Valentine

Colon definitions are great for designing scenes, and out of scenes you can build acts; out of acts you can build entire movies. The natural nesting structure of FORTH makes it perfect for animation. The script for "Circlove Life" was done right on the computer, after building a few special types of things such as loop constructs that control the range and parameters for the different dimensions. Simulation on the inside is animation on the outside.

The movie is made up of hearts. The definition of the heart fits on one screen. It's basically a loop that goes over a certain range that increments either by a positive or negative value depending on whether you're going up or down and uses the index of the loop to control everything from color to size to the spread. You can change the color table in a variety of ways. We had three different loops built out of CMOVES that were running at different frequencies. By picking harmonics of these, and getting different prime values, you could see the red move in, the blue move out, and you move your three color tables, for the three primaries, to go at relatively prime frequencies.

This not only gives you the building up of elements that you'd like. It also lends itself to the most readable code, and exploiting the possibilities of good FORTH style. You get to use the noun-verb idea, the definition of scenes. Extensibility is very important because all of this was all built very quickly out of something that had nothing to do with computer art, nothing to do with building movies, but basically a computer graphics language.

I suggest that those of you who are really concerned about the readability of FORTH, try doing something that is application-oriented, and try using the terminology of your application. The code is extremely readable.

*Howard Pearlmutter has spurred much activity in the area of computer graphics. He is author of a report for NASA titled* Interactive Computer Graphics: the Human Interface to Dynamic Simulation. *Howard is also the contact for Figgraph, the FORTH graphics special interest group. (408/425-8709)*

### Music

FORTH Dimensions *found Allen Strange, Professor of Music at San Jose State University, using FORTH in his campus synthesizer studio.*

We started about one year ago with an Ohio Scientific 6502 system procured on a small faculty grant to let us experiment with driving analog synthesizers with digital logic. I was interested in developing a language that could be used by anyone doing analog synthesis. We're not talking about commercial analog synthesizers

which are keyboard devices. Our studio is built around Buchla systems. Buchla was the first manufacturer in the early 60's to produce commercial synthesizers. These use all analog programming with the synthesizer itself.

We were interested in something that would breathe new life into analog synthesis. Everyone talks about digital synthesis and forgets about the special qualities produced by analog.

In the arts, the big problem when you're forced to learn a language is that it takes you further away from your art. In music your concept is translated into notes, then translated again into manuscript. Now the poor computer guy has to translate it one step further. But in FORTH we can call a process anything we want. You're back to being closer to your art. We decided FORTH was the way to go.

We've come up with a language called MASC—Metalanguage for Analog Synthesizer Control. It's a series of about 25 words that lets us do anything we want to do. For instance, we can cause the synthesizer to play a major scale, play a minor scale, play it backwards. We can randomize the notes in a major scale according to some format. We can create timing structures in musical terms, such as commands called WHOLE-NOTE and HALF-NOTE.

We can define sequences of things. For instance, we can either play in or type in a series of notes. Then we can read it out in a variety of ways: backwards, forwards, inside out, to generate musical events. The events are used to compile phrases; phrases are used to compile sections; sections to compile compositions. The language makes it easy for the performer to reorganize the formal aspects of his music. The performer can try putting this part first, then try putting it last, etc.

We're now using a Terak Computer, which is an RT/11-based system, a small PDP/11. We conned the campus computer center into giving us this thing, and in buying the FORTH package. The system uses 32 channels. The computer generates 8 channels of control output in the range 0 to 10 volts via D/As and 8 channels of timing pulses to initiate certain events. At the same time there are 8 channels of A/D input for controlled voltages, and another 8 channels of input dedicated to timing logic, to start and stop processes in the computer, count events, etc.

I've been teaching this class for a couple of semesters. Originally I tried teaching them this language without getting into FORTH. But we've made a few additions that make the FORTH more readable, so now I'm really teaching FORTH. One of the changes is that we implemented the word **INTEGER** instead of **VARIABLE**. **INTEGER** returns its value to the stack without having to say "@", and you can store into by saying "number TO name". This makes the description more natural. For instance, we'll continually refer back to a number that's used for timing information in a DO LOOP. If you want to change the tempo, you just say the new tempo TO the variable.

The students don't come out hotshot programmers, but they know what they're talking about. We've even managed to place a few students in industry working on video games where music is very important. It's unusual for a music major to graduate and be able to step into a relatively high paying job based on the relationship between music and technology.

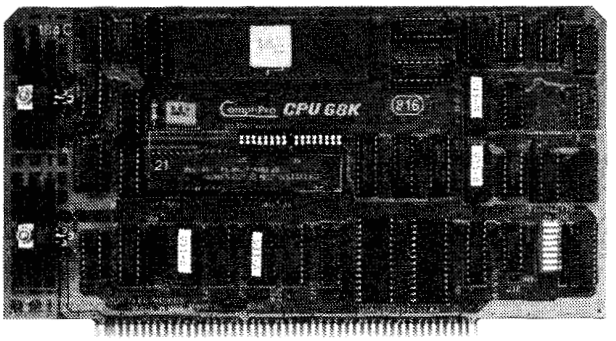
When we first started, we had about a hundred words. After doing this for two semesters we've thrown out most of these words because they conformed to too narrow an aesthetic. The danger of including powerful commands is that they imply an aesthetic. For a lot of people, the language was getting in the way. So we took it back to the bare bones.

These questions of aesthetics can be handled very well by FORTH just by using this core of words. The students use : and <**BUILDS DOES**> to generate their own aesthetic. MASC itself gives them only the bare tools to pass information back and forth with the computer. At first the students get mesmerized and think that the computer can do everything. But we're careful to teach them to use the computer only for what they need it for. Like when you need an extra set of eight hands. But making it complete enough to use for a complete performance is too much work, and

# Upgrade your Current CompuPro® System to a 68000 CPU and FORTH

## CPU 68K

### CPU 68K: A/T $695, CSC $850



**SPECIFICATIONS:**
The CPU 68K operates at 8 MHz assembled and tested (A/T), and 10 MHz for the Certified System Component (CSC), version of the board.

**FORTH HARDWARE REQUIREMENTS:**
CompuPro's DISK 1 floppy disk controller; INTERFACER 1, 2, 3, or 4; 64K of RAM and CPU 68K.

**COMPLETE FORTH OPERATING SYSTEM: $200. FEATURES:** an assembler, full screen editor, CP/M® file transfer utility, time-of-day/date stamping, shadow screen printing utility, and line editor, fully compatible with STARTING FORTH by Leo Brodie, as well as many other useful extensions.

All CompuPro products meet the most demanding mechanical and electrical standards, and are backed with one of the best warranties in the business (1 year limited warranty on all BOARD LEVEL products, 2 year limited warranty with exchange program for products qualified under our Certified System Component program). **Call CompuPro at (415) 562-0636 for additional information or to order.**

### COMPLETE 68K SYSTEM — $8995

## INCLUDES:

- **ENCLOSURE 2 DESK TOP**
- **8 MHz CPU 68K**
- **SYSTEM SUPPORT 1**
- **INTERFACER 4**
- **256K BYTES OF 16-BIT MEMORY**
- **1.5 MBYTES OF M-DRIVE/H**
- **DISK 1 CONTROLLER**
- **DISK ENCLOSURE WITH 2 QUME DRIVES (2.4 MBYTES)**
- **ALL CABLES**
- **mapFORTH & CP/M-68K™**

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

AUTHORIZED SYSTEMS CENTERS offer complete installation and implementation of our CPU 68K SYSTEM, Call (415) 562-0636 and ask us for the name of the SYSTEMS CENTER nearest you. **Price shown does not include dealer installation and support services.**

**CompuPro division Godbout Electronics — Oakland Airport, CA 94614**

not worth it. MASC could probably be critized for not taking care of certain things such as generating simple envelopes or random voltages. But the synthesizer itself does these things, so there's no need to submit that task to the computer. Still, the vocabulary is there to make these things, if necessary.

A keyboard makes strong aesthetic implications. Our synthesizer uses keys, but keys don't necessarily mean pitches. A keyboard is simply a voltage divider. We can use the keyboard to call up words. Say we have a word called **FUN1** which produces a bunch of random pitches and spins the sound around the room in our quadraphonic studio. (We can move sounds dynamically.) Then we have a word called **FUN2** which plays a C major scale in the right hand speaker and **FUN3** which plays a C major chord on another speaker. We put these words in an execution array. If we hit key 1, **FUN1** will play, but if we hit key 16, **FUN2** will play, will play, etc. So we can play the structure of a composition not by playing notes but by playing words.

The keyboard is a series of touchplates that know how hard you're hitting by reacting to body capacitance. So if you press the key hard it will spin around the room at one rate; press soft, it will spin at another rate. You decide what elements you want to control and what element you want to control it, then write the commands that recognize those actions. Over the last couple of years we've had over 50 people work on this. What we have is universal so it should be very useful to a lot of people.

*Note: The system described here will be taught in a two-week course starting on July 11, 1983. Contact Allen Strange, Music Department, San Jose State.*

### Multi-Media and Rock Promos

FORTH Dimensions *interviewed Peter Conn, President of Homer and Associates in Hollywood, California.*

We have two main computer systems that work together. One is a computerized optical printer interfaced to a paint system (raster graphics). The other is a 24-channel visual mixing console that controls 16 slide projectors and four 16mm projectors, as well as an audio tape machine with time code and music. The combination allows you to do a very complex mix of music and images in real time. FORTH Inc. was the only consulting firm who said "we can do that." We've been using FORTH for around two years now.

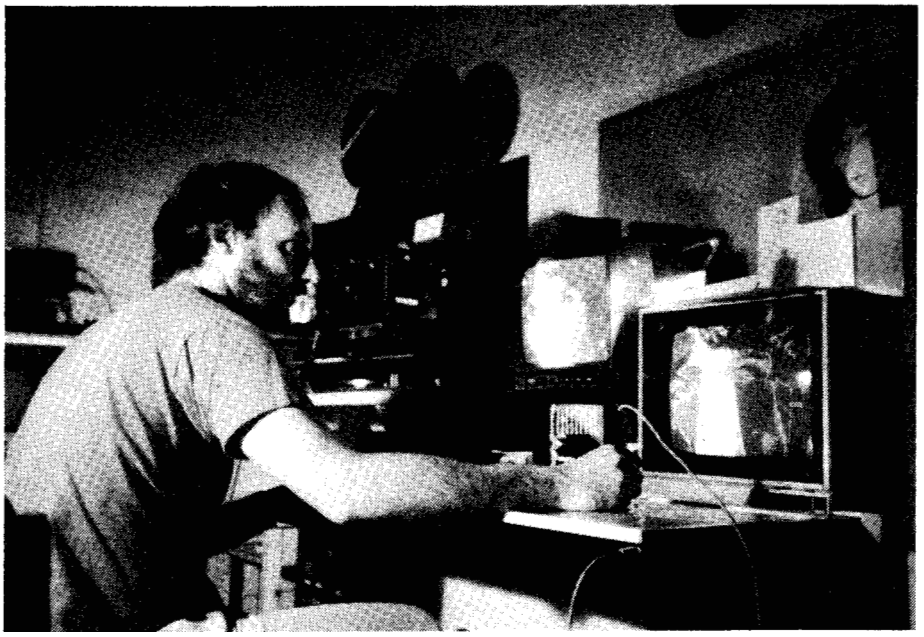
We recently bought a frame store, originally as a tool for aligning the optical printer. But the one we bought went far beyond what we needed because we got a good deal on it. Once Paul Rother, our programmer, got it working the software happened very fast. In a month we had a full fledged paint function, written completely in FORTH, with all kinds of features. It's 512 by 512 by 7 bits. We can paint to the RGB monitor, make slides, or movies that interface to the optical printer. Since then we've used the paint program on all of our rock promos, including the film we developed for Steve Miller's *Abracadabra.*

On that promo we shot actors on film, then fed it into the digitizer. Then frame by frame an animator would draw on top of the picture. The electronic graphics are later recomposited back with the original movie. It's a form of electronic rotoscoping. We created action in the frames that was not in the original; sparkles, enhancements, etc. In one scene there's a girl juggling scarves, lit by an overhead light. The animator tied her hand to the shadow of her hand on the floor with an electronic line as if it were a rubber band. That particular film has won awards all over the world. It just got nominated for best director American Video Awards 1982.

We just did one called *Atomic Dog* for George Clinton. We created a fictitious video game called *Atomic Dog*, using the computer graphics. Then we had a location that matched the game. The guy goes down into the game like TRON and the game comes alive with dancers. At the end of the four minutes he ends up back at the video arcade. You can see the live action doors, with the computer doors right next to them. We also have developed a three-screen multi-media show that will play in all the Six Flags

amusement parks across the country, and we did *The Great Rock and Roll Time Machine* which is playing at Magic Mountain right now. We developed these using the mixing console. Everything transfers to film, so the actual performance is all done on three synchronized projectors.

Developing our own paint system has saved us a lot of money. There are some products that are similar, but they're too expensive, in the $60,000 range. We've spent a lot less than that, and we have special features too. The best thing about our system is that it's programmable. You can sit there and change it, because it's in FORTH. Chuck Moore wrote the inital application on the mixing console. Paul Rother has done all the programming since then, and he's now part time. With their work, and with FORTH, I'm able to set up these complex scenes myself.



*Peter Conn, owner of Homer & Associates, sits at the electronic painting station of their computerized optical printer. From an album cover drawn by Andy Warhol, Peter is creating a scene for a rock video for Billy Squier. All software was written in FORTH by Paul Rother.*

ANNOUNCING

# THE JOURNAL OF FORTH APPLICATION AND RESEARCH

With the emergence of Forth as a powerful tool for applying computers, it's becoming difficult to keep up with its use in areas as diverse as laboratory and office automation, video games, and VLSI design. *The Journal of Forth Application and Research* will cover the exciting growth of the Forth community by providing a forum for users and researchers in science, industry and education. Each issue of the *Journal* will have refereed papers, technical notes, letters, book reviews, announcements and an index of current Forth related publications.

If you work with Forth, you need to know what your professional colleagues are implementing and why. If you are considering Forth, you need to know how and when to apply it. The *Journal* can be your focus on Forth.

## Subscriptions Vol. I

Vol.I will contain two issues, July and December 1983. Starting with Vol. II, each volume will have four issues.

| | | | |
|---|---|---|---|
| Charter Individual | $ 30 | Individual Sponsor | $100 |
| Company/Institution | $100 | Company Sponsor | $250 |

(Sponsors will be listed in each issue.)

Student rates available upon request. For subscribers outside North America, please add $10 for postage. Make checks or money orders payable to *The Journal of Forth Application and Research.*

## Information for Authors

Papers should describe applications or current Forth research, and must present original work not previously published. They should not be simultaneously under consideration for publication elsewhere. We also welcome review articles, short technical notes and announcements.

Application papers should include an instrumentation section, which describes the hardware and equipment used as well as the system configuration. Please give your Forth system's origin, and some definition of any major divergence from FORTH-79 or FORTH-83. Code should be well documented and relevent references should be cited. More specific manuscript guidelines are available upon request.

Please direct all manuscripts, inquiries, and subscriptions to:

The Journal of Forth Application and Research
P.O. Box 27686
Rochester, New York 14627

# 3–D Animation

*Paul Lutus and Phil Thompson*

*The following article is reprinted from a series titled "The Animated Apple With GraForth" originally appearing in* Softalk *Magazine.*

In any type of computerized 3-D graphics system, you start by creating a set of points, lines, and shapes in 3-D. Every point has a relationship to every other point: it can be higher or lower, closer or farther away, and more to the left or right. And, of course, this relationship depends on your point of view. The three different direction aspects of a point are represented using three numbers, or coordinates, labeled X, Y, and Z. It's the computer's job to convert your set of points according to some formula into points on a two-dimensional screen, using only X and Y coordinates. Then the points are connected with the appropriate lines, just as the 3-D points were connected with lines.

There are two different philosophies used in creating 3-D graphics. For the first, imagine a universe in which all of the 3-D objects exist. You describe the objects and tell the computer where they are in the universe. You then decide where your eye is, and which direction you're looking. The computer figures out which objects lie in that direction, converts them into a single two-dimensional image, and draws that image on the screen. This concept makes it fairly easy to represent complex scenes, but manipulating individual objects within that scene can be more time-consuming.

Another philosophy is to treat each 3-D object separately on the screen. You describe each 3-D object, then tell the computer where the objects should appear on the two-dimensional screen (or if they should appear at all), what size to draw them, and how they should be oriented. Each object is converted from three dimensions to two, independent of every other object. This means complex scenes can require more programming to produce, but manipulating each individual object is faster and easier. This is the technique used by GraForth.

GraForth allows you to manipulate 3-D objects through direct high-level commands. For example, the GraForth word *scale* sets the displayed size of a 3-D object, *xrot* rotates the object about the X axis, and *ypos* sets the vertical position of the object on the screen. These straight-forward commands provide an easy-to-follow method of generating 3-D graphics.

The 3-D process can be divided into two parts: first, the image is created using the *Image Editor* supplied on the GraForth system disk. Then, the GraForth commands are used to read the image and draw the object on the screen with the appropriate rotation and scale. The image may reside in any free area of memory and is not changed by the drawing commands.

Let's define a couple of words for this discussion. An "image" is a set of 3-D points and lines as stored in memory. An "object" is a picture of the 3-D image as it is manipulated and actually displayed on the screen. Images can reside in memory without being assigned as objects and drawn; and two objects, though positioned and oriented differently on the screen, can both use the same 3-D image in memory. (For example, two rotating cubes on the screen can use the same set of 3-D lines.)

For each image, the X, Y, and Z coordinates can range from -128 to 127, giving a possible 256 positions along each of the three axes , which is plenty for most applications. The actual number of lines in an image is limited only by the amount of available memory. (Each end point of line entry in the image uses four bytes of memory.)

Up to sixteen different objects can be manipulated at one time in GraForth. They are numbered 0 through 15, and referenced with the GraForth word *object*. After giving an object command, the 3-D commands will manipulate that object until another object command is given. For example, if you type:

```
3 OBJECT
30 XROT
10 SCALE
```

then object 3 will be rotated 30 units around the X axis and scaled to a size of 10. To manipulate a number of objects, you select each object in turn with *object*, then give the appropriate commands for that object.

Here is a quick summary of the individual 3-D commands, their effects, and the appropriate ranges of numbers to use:

**Xpos, ypos.** These set the X and Y position on the screen of the 3-D point (0,0,0) for the object and are used for positioning the object in the appropriate place on the screen. Xpos can range from 0 to 255 and ypos can range from 0 to 191. At the extremes, however, the object may overlap the edge of the screen, causing wrap-around.

**Scalx, scaly, scale.** These commands determine the size of the object on the screen. Scalx sets the width and scaly sets the height. The word *scale* simply sets both width and height to the same number simultaneously. The range is from -31 to 31. A scale of 0 produces a displayed object with no thickness, and negative numbers create a mirror-image effect. Since two objects can use the same image in memory, symmetrical objects, such as bird wings, can be created using two objects side by side, with positive and negative scale numbers.

**Scalz.** This determines the amount of perspective used. Perspective is what causes the front of an object to appear larger than the back. A large perspective number makes the front a good deal larger, and negative numbers provide "reverse perspective," with the back of the object larger than the front. Zero perspective means the front and the back will be the same size. The range, as above, is -31 to 31.

**Xrot, yrot, zrot.** These commands rotate the current object around each of the three 3-D axes. A complete rotation is divided up into units from 0

to 256. Zero is no rotation, 64 is a right angle, 128 is the same as 180 degrees, and 192 is three-quarters around the circle. Values greater than 256 or less than 0 can also be used for rotating more than once around. For example, a rotation to 258 units is the same as to 2 units.

Note: The actual rotation of the object changes for every other rotation value. This means that if you rotate an object in steps of 1 unit per draw, the view of the object will change every other draw, making the animation appear slower. It's best to increment rotation values in steps of 2.

**Xtran, ytran, ztran.** These commands translate, or "slide," the object in each of the three directions in space. The object can be shifted as long as none of its points falls out of the -128 to 127 position range. If this happens, a wraparound effect will occur. Therefore, translation works best with small images, having room to move.

**Objcolor.** This determines what the object's color will be when it is drawn if color was not specified when the image was created. If color was specified, then objcolor is ignored. The standard GraForth color numbers (1 through 7) are used. Note that objcolor also sets the normal color command, so be sure to reset color to the desired value after using objcolor.

Table 1 shows the 3-D parameters and the range of values they use. You can experiment with the definitions by changing some of the parameters from the editor and recompiling. Of course, you can also type the word definitions directly into GraForth from the keyboard.

Creating animations with GraForth's 3-D graphics is easy and straightforward. As we mentioned in an earlier column, animation is simply a series of still pictures displayed rapidly one after another, providing the effect of movement. One fast way to generate this movement is with a do-loop:

```
257 0 DO I YROT DRAW 4 + LOOP
```

This example rotates the object a full circle around the Y axis. Since the loop is in steps of four, it repeats 64 times, producing 64 separate draws, one after another. For each draw, the rotation around the Y axis is set to the loop value, incrementing from 0 to 256.

This type of animation is straightforward, but for most applications a number of parameters need to be manipulated at once. Let's look at how to do more complicated manipulations with a few examples.

When using a do-loop, usually one draw will be performed each time through the loop. The size of the loop then determines how many times the object will be drawn. To change the parameters, two approaches are possible: the loop value can be used to generate the desired parameter values, or separate variables can be used to keep track of each parameter.

In the first method, the conversion from loop value to parameter value is done with short formulas. For example, if you want the tetrahedron to rotate around the Y axis three times for each rotation around the X axis, you can use this routine:

```
: THREE.ROT
  257 0 DO
    I XROT
    I 3 * YROT
    DRAW
  2 +LOOP ;
```

After entering *three.rot* into the editor and compiling (or entering it directly from the keyboard), it can be run by simply typing:

```
THREE.ROT
```

The trick is to find the right formula for the desired motion. Suppose, with the above example, you also wanted to make the tetrahedron grow in size from 12 scale to 20 scale. The change from 0 to 256 in the loop must be translated to change from 12 to 20. Note that the difference between the start and end loop values is 256, and the difference in the scales is 8. If we divide the loop value by 32, we get a range of 0 to 8. If we then add 12, we get the desired range of 12 to 20:

```
Loop value 0 /32 = 0 ... 0 + 12 = 12
  Scale value
Loop value 256 /32 = 8 ... 8 + 12 = 20
  Scale value
```

The new routine looks like this:

```
: ROT&SCALE
   257 0 DO
   I YROT
   I 3 * YROT
   I 32 / 12 +  SCALE
   DRAW
LOOP ;
```

| Parameter | Range | In steps of |
|---|---|---|
| XPOS | 0 to 255 | 1 |
| YPOS | 0 to 191 | 1 |
| SCALX | -31 to 31 | 1 |
| SCALY | -31 to 31 | 1 |
| SCALE | -31 to 31 | 1 |
| SCALZ | -31 to 31 | 1 |
| XROT | 0 to 255 | 2 |
| YROT | 0 to 255 | 2 |
| ZROT | 0 to 255 | 2 |
| XTRAN | -128 to 127 | 1 |
| YTRAN | -128 to 127 | 1 |
| ZTRAN | -128 to 127 | 1 |
| OBJCOLOR | 1,2,3,5,6,7 | |

Table 1.

Now we'll look at a program adapted from the "rolling tetrahedron" display in the GraForth demonstration program. The tetrahedron moves down and to the right, rotates end over end, and grows and shrinks, giving the appearance of rolling closer, then farther away. You can use this routine with any image in memory.

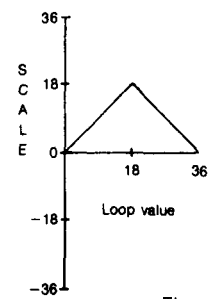



Figure 1.

```
: ROLL.OBJECT
  37 0 DO
    I 3  * XROT
    I 5 * YROT
    I 6 * 25 + XPOS
    I 3 * 35 + YPOS
    I 18 - ABS CHS 18 + SCALE
    DRAW
  LOOP ;
```

None of these formulas were arrived at by magic. As the routine was written, we tweaked each formula until we got the desired display. Here are the numbers that come out:

```
Loop value: 0 to 36
XROT: 0 to 108
YROT: 0 to 180
XPOS: 25 to 241
YPOS: 35 to 143
SCALE: 0 to 18, then back to 0
```

The scaling formula deserves more comment. The desired effect was to have the object grow and then shrink.

We could have used two scaling loops one after another—the first increasing and the next decreasing. But then we would have had to keep all the other parameters moving smoothly through the transition from one loop to the next, without a skip in values. For simplicity, we decided to use a single loop.

With the loop value moving from 0 to 36, we wanted the scaling function to slide from 0 to 18 and back to 0. This can be shown with Figure 1. Figure 2 shows the steps we used to achieve the effect. Sometimes a more complicated animation cannot be performed inside a simple do-loop. This is especially true if the user is interacting with the program through a joystick or keyboard, and the program must make decisions. In this case, it's often best to use separate variables to keep track of each parameter. The parameters can then be updated at any time from the running program. The following program duplicates the *Roll.Tetra* routine using this technique.

```
VARIABLE XR (X rotation)
VARIABLE YR (Y rotation)
VARIABLE XP (X position)
VARIABLE YP (Y position)
VARIABLE SC (Scale)
VARIABLE DIR (Scale direction
larger or smaller?)

: UPDATE.TETRA
  XR 3 + DUP -> XR XROT (increase X
rotation by 3)
  Y R5 + DUP -> YR YROT (increase Y
rotation by 5)
  XP6 + DUP-> XP XPOS (increase X
position by 6)
  YP 3 + DUP -> YP YPOS (increase Y
position by 3)
  DIR IF (if scale is increasing:)
  SC 1 + DUP -> SC SCALE (increase
scale by 1)
  SC 18 = IF 0 -> DR THEN (change
direction?)
  ELSE
  SC 1 - DUP -> SC SCALE (Decrease
scale by 1)
  THEN ;
: ROLL.TETRA1
  0 - >XR 0 -> YR (initialize
variables)
  25 ->XP 35 -> YP
  0 ->SC
  1 -> DIR (set scale direction)
  DRAW (Draw first object)
  36 0 DO (Start loop)
  UPDATE.TETRA (Set new
parameters)
  DRAW (Draw object)
  LOOP ; (Loop back)
```

We used a do-loop to run the animation since no branching decisions were needed for this program. If they were required, the current value of any 3-D parameter would always be available.

For smooth animation, the GraForth 3-D graphics routines automatically take advantage of both hi-res screen pages in the Apple memory. During 3-D animations, one screen area is displayed while the other is being invisibly updated. This way, the lines are not shown being erased and redrawn. This is only true for 3-D graphics. GraForth text printing, line drawing, and character graphics always draw to both screens simultaneously. In this way, the screen-flipping 3-D graphics can be mixed with other kinds of graphics without causing lines and characters to repeatedly appear and disappear.

The sequence GraForth uses in putting a 3-D object on the screen is a four-step process: whenever the word *draw* is executed, the drawing routines are first directed to the graphics screen that is not currently being displayed. Then the previous 3-D objects are individually erased line by line by following the parameters that were originally used to draw them. Next, the new objects are drawn on the screen using the current parameters. Lastly, the display is switched to this screen, so that the new objects can be seen.

To increase speed, the word draw only works with the objects that have been referenced since the last draw command. This reference can be made by giving the object one or more new parameters, or by simply calling it again with *object*. This means that objects that don't need to be changed can be left on the screen as they are and will not slow the drawing of objects still in motion.

Suppose you're manipulating two 3-D objects (call them objects 1 and 2) simultaneously. First, both of them are in motion, and the animation toggles between the two graphics screens with each draw command. Then you decide to stop the motion of object 1, while continuing object 2. To do this, you simply stop giving object 1 any new commands. Since object 1

*(Continued on bottom next page)*

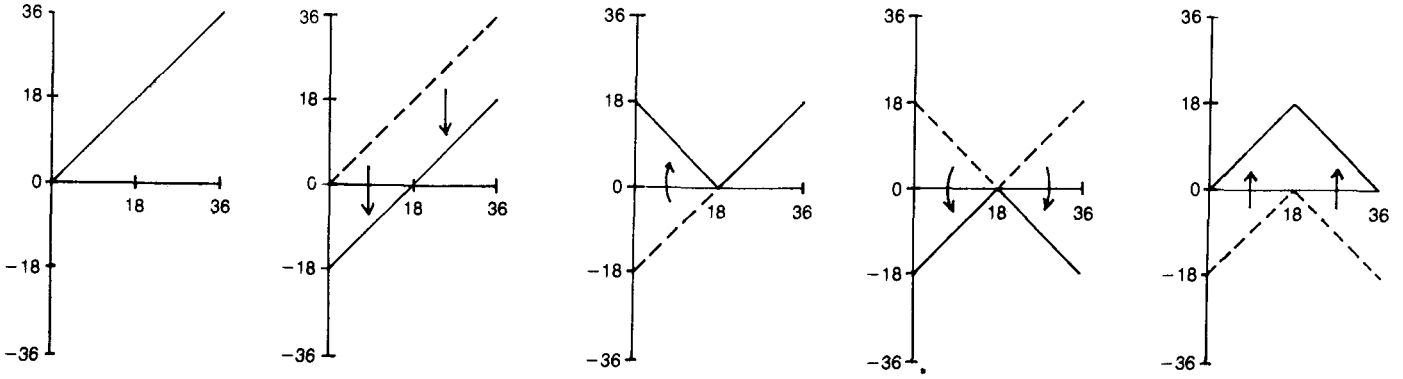



Figure 2.

# Space Graphics Problem

The Mullard Space Science laboratory, part of the Department of Physics and Astronomy of University College, London, is currently running polyFORTH on a Motorola EXORset 33 to tackle the problem of displaying and storing data produced by its latest generation of imaging detectors.

The detectors will be used on future space missions to record X-ray images of distant astronomical objects, and also at ground based observatories where their extreme sensitivity will allow observations of the faintest and farthest known galaxies in the universe.

A desktop microcomputer system was chosen for the task since it could meet most of the system requirements at a much lower cost than commercial systems, and special features could be included by writing the necessary control software. The chosen computer was the Motorola EXORset 33, based on the MC6809 microprocessor. To enable fast software development the interactive, high-level language polyFORTH was selected.

For detector development or ground-based use, each individual photon registered by the detector has its coordinate recorded in the memory of the EXORset, allowing an image to be built up over a period of minutes. The multi-tasking capabilities of polyFORTH allow an image display to be generated on the graphics screen and updated in real time with an overlay of numerical data, which is also frequently updated, while the data from the detector is handled on an interrupt basis. The operator may also enter commands via the keyboard function keys during the accumulation of an image.

The image data may be displayed in a variety of ways, such as a "contour" 3D projection, or as a "grey-scaled" image by assigning a different number of dots to each pixel according to its brightness. When a satisfactory image has been accumulated the data may be saved on floppy disc, and transferred to a minicomputer for detailed image processing if required. Since the build up of the image can be monitored in real time, the accumulation of useless data can be stopped at an early stage, reducing wasted time.

This development would have been very difficult using the EXORset's BASIC, not least because the BASIC interpreter uses the graphics display memory, making interactive development of graphics impossible! polyFORTH and its graphics package however fit in about 12K bytes, leaving plenty of room for graphics display, data, and the application program.

*This article is reprinted from a periodical published by Comsol Ltd., England.*

---

*(3-D Continued)*

was previously in motion, the picture of the object on the two graphics screens is different. As the animation continues with object 2, the display will switch back and forth between the two screens. The two pictures of object 1 will alternate back and forth, rather than remaining still.

The solution to this problem is simple: when you don't need to move an object any more, give it one extra *object* command, without any new parameters:

```
1 OBJECT
```

This will cause the same picture of the object to be drawn on the second graphics screen. The two pictures of the object will then be identical, and the object will remain still while other objects are manipulated.

**Moving Faster.** With a little extra planning, the speed of 3-D graphics can often be increased considerably. The line-by-line undrawing of each 3-D object uses as much time as drawing the new object. A faster method to remove old images is simply to erase the area of the screen the object lies in, and then not to bother doing a line-by-line erase.

The GraForth word *undraw* is designed for doing just this. Undraw erases a portion of the screen just as unblk does, on a character-size basis. However, undraw also sets a flag telling GraForth not to do a line-by-line erase of the 3-D object. After setting the block size and the position appropriately, you can erase the object yourself, so that the 3-D routines don't have to erase it. This method requires that you know what rectangular area of the screen is used by the object and that no other graphics line in this area, since they would also be erased.

Here is an example of using undraw. Starting from scratch, let's first get an object onto the screen: **0 40 18 24 WINDOW ERASE** (Optional)

```
CR 132 PUTC PRINT " BLOAD
CUBE,A2816 " CR
OBJERASE
0 OBJECT 5 SCALE
20 XROT 20 YROT
DRAW
```

An easy way to determine the block size and placement to use with undraw is to fill the screen with characters, then draw the object over them:

```
0 VTAB 1000 0 DO I 10 MOD . LOOP 0
OBJECT DRAW
```

By simply counting down and across, you can see that the cube fills a block nine characters wide by eight characters tall, starting at *8 vtab 14 htab*. The undraw command can be used to erase this block during a 3-D animation:

```
ERASE
9 8 BLKSIZE
```

Now type this entire line, and then press the return key:

```
8 VTAB 14 HTAB 257 0 DO
I YROT UNDRAW DRAW 4 +LOOP
18 VTAB
```

This sets the character position for the block and rotates the object while erasing the block with undraw. Compare it with the same loop without undraw:

```
257 0 DO I YROT DRAW 4 +LOOP
```

The difference is quite noticeable.

# Double-Precision Math Words

*L.H. Bieman*
*Energy Development Associates*
*(a Gulf + Western Company)*
*Madison Heights, Michigan*

Convinced of the beauty of programming in FORTH, I started an application to interface a lab experiment with a microprocessor. (Another factor was that FORTH was the only language available besides Assembler and I am not fond of programming in Assembler.) My scientific background yearned to use floating-point math, but I decided to stick with the credo of FORTH programmers: use integer arithmetic; it's fast and exact. Unfortunately, my application required the use of double precision numbers for which there were very few math operations in my version of FORTH.

There were four basic operations available: double precision addition subtraction; multiplication of two singles to form a double; and division of a double by a single, yielding a single. This was not enough for my application. I needed to take the ratio of two double precision numbers, scale double precision numbers, etc. Thus, I did what a FORTH programmer must, create the needed words. Hopefully these words, which are found on the three screens shown, will help others who have a version of FORTH with limited double precision math.

The math words for signed numbers include:

**D/** Takes the ratio of two double precision numbers to yield a single precision quotient.

**D*S** Multiplies a double times a single precision number to yield a double precision product.

**D/S** Divides a double by single precision number to yield a double precision quotient.

```
120 TRIAD


SCR # 120
  0 ( DOUBLE PRECISION MATH WORDS - BIEMAN )
  1 : 2SWAP  ( D2,D1 --- D1,D2 )
  2    ROT >R ROT R>  ;
  3 : OVER2  ( N3,N2,N1 --- N3,N2,N1,N3 )
  4         >R OVER R>  SWAP  ;
  5 : D/SIGN  ( D2,D1 --- D2,D1,NSIG )
  6    OVER2 OVER  XOR   ;
  7 : D/  ( D2,D1 --- NQUOTIENT )
  8    D/SIGN >R     ( FIND SIGN AND STORE )
  9   DABS DUP 1+ DUP >R U/ SWAP DROP
 10    ( BREAK DIVISOR INTO N1*N2 )
 11  >R DABS R>   U/ SWAP DROP ( DIVIDED BY N1 )
 12  0  R>   U/ SWAP DROP      ( DIVIDED BY N2 )
 13    R> +-  ( PUT SIGNON QUOTIENT)  ;
 14    ;S
 15


SCR # 121
  0 ( DOUBLE PRECISION MATH WORDS - BIEMAN )
  1 : T*  ( UD,UN --- UT )
  2    DUP ROT U* >R >R ( MULT UPPER PRECISION PART )
  3        U*           ( MULT LOWER PRECISION PART )
  4    0 R> R>  D+      ( ADD BOTH PARTS ) ;
  5 : T/  ( UT,UN --- UD )
  6    >R  R  U/ SWAP  ( DIVIDE UPPER PRECISION PART )
  7   ROT 0 R U/ SWAP  ( DIVIDE LOWER PRECISION PART )
  8   ROT  R> U/ SWAP DROP ( DIVIDE REMAINDER )
  9   0 2SWAP SWAP D+ ( ADD PARTS )   ;
 10 : U*/  ( UD,UN,UN --- UD )
 11    >R T* R> T/ ;
 12      ;S
 13
 14
 15


SCR # 122
  0 ( DOUBLE PRECISION MATH WORDS - BIEMAN )
  1 : SA  ( D,N --- DABS,ABS,SIGN )
  2   2DUP XOR >R  ( FIND AND SAVE SIGN )
  3   ABS >R  DABS R> R>  ( CHANGE TO ABS ) ;
  4 : D*S  ( D,N --- D )
  5   SA >R  T* DROP R> D+- ( MULT DOUBLE TIMES SINGLE ) ;
  6 : D/S  ( D,N --- D )
  7   SA >R 0 SWAP T/ R> D+- ( DIVIDE DOUBLE BY SINGLE ) ;
  8 : D*/  ( D,N2,N1 --- D )
  9    >R SA R> DUP ROT XOR  >R ABS ( FIND SIGN AND ABS )
 10    U*/    R> D+-  ( CALCULATE AND SET SIGN )  ;
 11     ;S
 12
 13
 14
 15


  ok
```

**D*/** Takes a double precision and multiplies and divides by single precision numbers, yielding a double precision (very useful for scaling).

The math words for unsigned numbers include:

**T*** Multiplies a double times a single precision number to yield a triple precision product.

**T/** Divides a triple by a single precision number to yield a double precision quotient.

**U*/** Takes a double precision and multiplies and divides by a single precision number, yielding a double precision.

In the comments found on the screens, U stands for unsigned, N stands for single precision, D stands for double precision and T stands for triple precision. For some versions of FORTH the **R** must be replaced with **R@**. Another item to check is that the **U/** works correctly for divisors greater than hex 7FFF.

There are some modifications of the words that might suit your application better. You might want **D/** to produce a double precision quotient rather than single precision. This can be done by: compiling **D/** after **T/**, changing line 11 on screen 120 from **R> U/ SWAP DROP** to **0 R> T/**, changing line 12 from **U/ SWAP DROP** to **T/**, and changing line 13 from **+-** to **D+-**. Of course, the execution time for **D/** will greatly increase. You might want to save the remainder in **T/** by changing line 8 on screen 121 from **SWAP DROP** to **>R** and changing line 9 from **D+** to **D+ R> ROT ROT**. If you make this change, don't forget to change **T/** to **T/ ROT DROP** in **U*/**, in **D/S**, and (if it appears) in **D/**.

*ACKNOWLEDGMENT*

*I would like to thank Jim Galloway for introducing me to FORTH and for being kind enough to review this note.*

FORTH Dimensions *requested our resident math operator authority (and Standards Team Secretary) Robert L. Smith, to review Mr. Bieman's contribution. Mr. Smith writes:*

I implemented and tested the functions described in this article and found that they worked pretty much as advertised, and even better than expected. I expected that the **D/** routine would fail because the factoring of the double precision divisor would yield two single precision numbers whose product would not give the original double precision value. That turns out to be not as important as I assumed because of the order in which the partial quotients are taken. It is likely, however, that the quotients are just slightly more inaccurate than one would obtain from a very precise technique. I was surprised also at his suggested method of obtaining the double precision quotient of two double precision numbers. However, if the denominator has a high order part of zero, then the quotient will be OK, with proper full precision. If the most significant part of the denominator is non-zero, then the quotient will have a zero in its most significant part. The greatest error will be just one or two least significant bits.

The routines are written in FORTH, so some small changes have to be made to convert to 79-Standard. The function names are somewhat new to me, but there is not sufficient widespread use of these sorts of functions to say that any names are traditional. I believe that routines of this sort are useful, since FORTH's traditional 16 bit precision is insufficient for a number of mathematical functions, and even for some measured physical quantities. In my opinion FORTH badly needs a good library of useful mathematical routines in the public domain. The routines here could possibly be a small part of that library.

I was asked if the routines are optimal. They appear to give approximate results in a much shorter time than a more precise method would. There seems to be only trivial improvements that could be made to get the same results. This paper has some rough gems that others might be able to polish. *—Robert L. Smith*

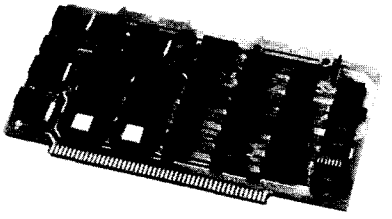
# DEVELOPMENT TOOLS

**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 – includes our new tools diskette

**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

**JULY 11-15 SEPT. 12-16**
**OCT. 17-21 OCT. 31-NOV. 4**

$395 Incl. Text

### Advanced FORTH System & Tools

- FORTH Tools
- Engineering Applications
- Floating Point
- Communications
- Sorting & Searching
- Project Accounting System
- Process Control
- Simulations
- FORTH Internals
- Assemblers and Editors
- Other Compilers
- Cross-Compilation Theory
- Romability, Multitasking, Timesharing
- Task Scheduling Algorithms

**SEPT. 19-23**
**NOV. 14-18**

$495 Incl. Text

Instructors: 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

# Add a Break Point Tool

Here's a scenario familiar to most programmers: you have a large section of an application running, but somehow right in the middle of things something bizarre happens—something that "couldn't possibly" be happening. The application either crashes mysteriously, or continues past the problem, too late for you to figure out where things went wrong.

Here's a very useful, ingenious debug tool that lets you halt your application in mid-operation, then use the full power of FORTH's interpreter to examine the stack, contents of variables and so on, and then resume. Before I heard about this tool, I was prone to editing a **QUIT** into a suspected spot in my code to terminate the application, then poke around for clues. The technique described in this article still lets you poke around, but from within the temporarily suspended application! (Those of you with multiprogrammed FORTH systems and multiple terminals know how useful this can be.)

The technique (originally called **PAUSE GO**) was invented by Frank Seuberling, adapted here by Kim Harris and brought to my attention by John Clark. The idea is simple: you edit the word **BREAK** into your aplication where you want the breakpoint, then recompile. When your application executes **BREAK**, you enter a special interpreter.

You may type any normal FORTH commands and press return. After completion of your commands, the special interpreter will respond "aok" instead of "ok" to remind you which interpreter it is, then await more commands. You can stay in this special interpreter as long as you want.

When you're ready to have the application resume, enter the word **GO** and press "return". **GO** will exit the special interpreter and allow the definitions that had been executing to resume (their addresses had been nested and patiently waiting on the return stack all along).

The real work is done by the **BEGIN** ... **AGAIN** loop in **BREAK**, which obviously is the special interpreter, and the phrase **DROP R> DROP** in the word **GO**, which unnests two levels: from **EXECUTE** inside **INTERPRET**, and from **INTERPRET** inside **BREAK**.

That's all the code you really need except for one aspect of "human factors." If **BREAK** consisted only of this **BEGIN AGAIN** loop, then if you were to cause an abort while inside the **BREAK** interpreter (by misspelling a word or underflowing the stack, etc.) **ABORT** would call **QUIT** which would cause an immediate cessation not only of the **BREAK** interpreter but of the suspended application as well.

At that point, if you don't happen to notice you're back in FORTH's interpreter and you type **GO**, you'll crash. To prevent such an occurrence from blasting you off to never-never land, a "check" has been added—the value of the return stack pointer is saved at **BREAK** time, and checked at **GO** time. If they don't match, you've changed levels, and an error message will result.

Another pleasant addition has been the stack dumps—both data stack and return stack—on line 6. **BREAK** will run fine without either of these if you haven't got them yet.

Kim Harris has noted a possible enhancement in the margin of the listing. By saving the contents of the return stack at break-time in an array, then restoring the return stack to its former condition at **GO**, then you could resume even without having to be in the special interpreter. Come to think of it, you wouldn't even *need* a special interpreter!

The possibility of suspending one application in mid-stream, performing any number of other tasks, then resuming at your leisure is very interesting for a non-multiprogrammed environment.

*—Leo Brodie*

```
Screen # 60
  0 ( Debugging tools: BREAK & GO    WF    13DEC81 KRH )
  1 ( by Frank Seuberling, 5/4/81 )
  2 VARIABLE CHECK    ( CHECKs if Rstack changed since BREAK )
  3     ( contains RP@ at time of BREAK )
  4
  5 : BREAK    ( - ) ( compile BREAK into : def )
  6     CR   ." BREAK S= "    S.    CR ." R= "   R.N
  7     RP@   4 -   CHECK !    0 BLK !
  8     BEGIN    QUERY    INTERPRET    ." aok" CR    AGAIN    ;
  9
 10 : GO    ( - ) ( enter GO to resume )
 11     RP@   CHECK @   = IF    R> DROP    R> DROP
 12     ELSE    ." can't resume "    QUIT    THEN    ;
 13
 14
 15
```

# Extending the FORTH Compiler

*Luke Seeto*
*Christchurch, New Zealand*

The following discussion deals with extensions to the FORTH compiler. A familiarity with the FORTH compiler and its words is required for implementation. Readers are referred to the following manuals.

*—fig-FORTH installation Manual*
*—Assembly Language Source Listing of fig-FORTH*

A description of the FORTH system used is first discussed.

FORTH can be implemented for both a development environment and a target environment. Such a dual environment has

- separate RAM memory for variables.
- separate compiler/operating system memory.
- separate program code memory; this is RAM in a development environment and ROM in a target environment.
- separate dictionary memory.

In our FORTH systems 8K byte blocks of memory at 8K boundaries can be bank selected by controlling any of eight bank-select registers to allow any memory address to be repeated up to 32 times. Figure 1 shows the memory map used which is as follows.

RAM memory for variables is at locations $0000 - $1FFF ($ stands for hexadecimal). This can be extended to location $5FFF for the target environment only. RAM memory also exists at locations $4000 - $5FFF as a number of selectable banks of 8K bytes, except for the first bank where the compiler/operating system resides. RAM memory residing in banks is handled transparently as is RAM at locations $0000 - $1FFF; this is the case also for the development environment even though the compiler resides in the same address space.

Locations $2000 - $7FFF for the first bank contains the compiler/operating system and input/output memory space. Into this area is placed source compiling code which is not required for ROMing. The compiler is extended as required.

Program code resides at locations $8000 - $FFFF for the first bank. Program code also resides at locations $8000 - $9FFF as a number of selectable banks of 8K bytes.

Dictionary memory space exists at locations $A000 - $BFFF as a number of selectable banks of 8K bytes except for the first bank where program code resides. The dictionary is transparent to the user and is only for the development environment.

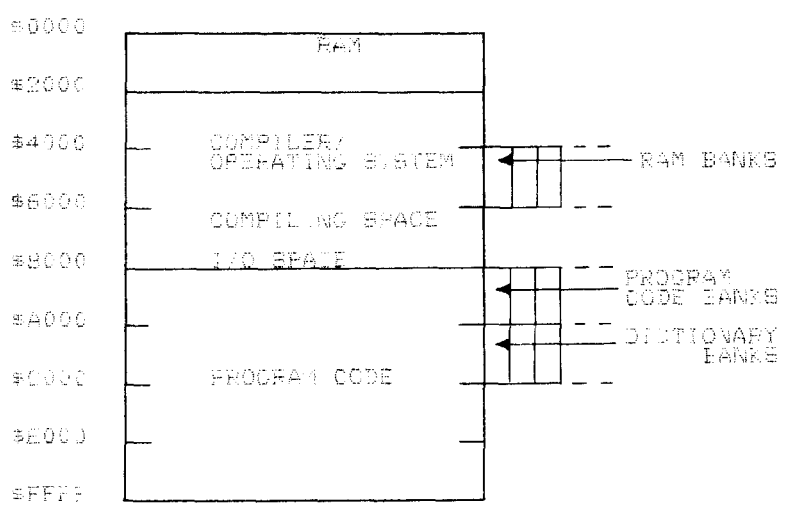



**FIGURE 1 - MEMORY ORGANIZATION**

Why use such an unusual system? Essentially to extend the memory space past 64K bytes. This has the added advantage that no cross-compilation is required to produce ROMable code and a huge dictionary space is available. The dictionary is fast when accessed and can periodically be culled. With ample dictionary space, names can be meaningful rather than cryptic and the compiler can be extended to relieve the user's load.

**Number Base**

Numbers in FORTH can be expressed in decimal or hexadecimal base. Using the words **DECIMAL** and **HEX** establish the base of succeeding numbers in the source program.

Hexadecimal numbers can also be entered as characters 0–F preceded by a $ sign by modifying the compiler as follows:

- define a variable **IBASE** similar to **BASE**.
- **NUMBER** is to first calculate **IBASE** as 16 if the first character of the Word is a **$** sign; otherwise it is as for **BASE**.
- **(NUMBER)** is to use **IBASE** rather than **BASE**.

A listing from the compiler/operating system is shown (see listing 1). Any user program which calls **NUMBER** will remain correct. Any which calls **(NUMBER)** will not; in which case change **(NUMBER)** to **(INUMBER)** in the listing. Also the Word **$** is specifically tested for; this is because interpretation first mis-matches against the dictionary before doing a number conversion and not vice-versa as in some FORTH systems.

Typically, use of an implicit hex number scheme avoids confusion.

**Data Structure**

Normally program code can be structured such that only the contents of a data structure, and not the memory address of the data, need to be referenced. In particular the Words **C@ CI CSETO CINCI ... @ I SETO +I** ... can be eliminated from a piece of program code.

Thus it is possible in FORTH to say **A? 1 + A=** to correspond to the BASIC statement **LET A = A+1**. Typically citing **A?** and **A=** generates less

program code and runs faster. Also, the source program is easier to follow without excessive manipulation of the stack. The extra cost is the dictionary space. Citing **A?** and **A=** is independent of whether the structure is a byte or a word variable. Hence the statement **A? 1 + A!** and not **A INC1**.

The data structure described below allows a user to define a named data, and automatically generated are its fetch/access structure and its store structure which are cited by name. All access to the data structure by name are re-entrant.

The following data structures are an alternative to the conventional variable data structure in FORTH; namely **VAR/VARIABLE CVAR ARRAY**.

**word data structure**

Variable space of 16 bits is allocated with the data structure

**M-VAR A**

This is equivalent to the statements

```
VAR A$
: A? A$ @ ;
: A= A$ ! ;
```

**byte data structure**

Variable space of one byte is allocated with the data structure

**M-CVAR A**

This is equivalent to the statements

```
CVAR A$
: A? A$ C@ ;
: A= A$ C! ;
```

**array data structure**

Variable space of n bytes is allocated with the data structure

**n M-ARRAY A**

This is equivalent to the statements

```
n CONSTANT A[L]
A[L] ARRAY A[$]
: A[?] A[$] + C@ ;
: A[=] A[$] + C! ;
```

Thus to fetch/access the fifth element of the data structure, use **4 A[?]**. The limit is also defined by name for use with DO loops.

**implementation details**

A description of how **M-VAR** can be implemented is given. **M-CVAR** and **M-ARRAY** are similar.

Firstly, program code should be generated together to optimize

LISTING 1: NUMBER CONVERSION USED BY COMPILER/OPERATING SYSTEM

```
                                                    VS2.0  JAN 83    PAGE- 1

SCR..1    0 NOTE      LISTING FOR NUMBER BASE
          1
          2           $24 IS HEX 24 FOR ASCII $
          3           IBASE? IS IBASE @
          4           IBASE= IS IBASE !
          5
          6           ----------------------.
          7
          8           ->

SCR..3    0 NOTE      ENTRY DOES NOT CONFORM TO FORTH SYSTEMS.
          1
          2
          3
          4 : (NUMBER)
          5           0 SWAP                          ( START VALUE = 0     )
          6           BEGIN 1+                        ( NEXT POSITION       )
          7                 DUPLICATE C@ IBASE?       ( CONVERT A DIGIT     )
          8                 DIGIT                     ( VALID DIGIT?        )
          9                 IF  >R                    ( - YES,              )
         10                     SWAP IBASE? *          (   ACCUMULATE        )
         11                     R> + SWAP              (   SUCCESSIVELY      )
         12          WHILE                           ( REPEAT UNTIL STRING )
         13          ;                               ( IS TERMINATED BY A  )
         14                                          ( NON VALID DIGIT     )
         15          ->

SCR..5    0
          1
          2           NOTE      ****************************************
          3                                                             *
          4                     CONVERTS A CHARACTER STRING TO A        *
          5                     NUMBER VALUE,                           *
          6                     EITHER USING CURRENT BASE               *
          7                        OR IMPLICIT (TEMPORARY) HEX BASE     *
          8                                                             *
          9                     FORMAT IS                               *
         10                       OPTIONAL LEADING - SIGN              *
         11                       FOR CURRENT BASE,                    *
         12                       LEADING $ SIGN FOR IMPLICIT HEX BASE *
         13                                                             *.
         14
         15                                  ->

SCR..6    0           NOTE                                            *
          1                     QUIRKS ARE                              *
          2                       ONLY LAST 4 DIGITS OF HEX/IMPLICIT HEX*
          3                       BASE IS RELEVANT,                     *
          4                                                             *
          5                       OVERFLOW NOT CATERED FOR DECIMAL BASE *
          6                                                             *
          7                                                             *
          8                                                             *
          9                       ON ENTRY,                             *
         10                          PSW1 = START ADDRESS OF STRING     *
         11                       ON RTN,                               *
         12                          PSW1 = CONVERTED VALUE             *
         13                          ABORTS IF NOT A VALID NUMBER       *
         14                    ****************************************.
         15                                  ->

SCR..7    0 : NUMBER
          1           DUPLICATE 1+ C@                 ( GET FIRST CHARACTER )
          2           DUPLICATE $24 =                 ( = $ SIGN ?          )
          3           IF    DROP                      ( - YES,              )
          4                 DUPLICATE C@ 1 =          (   WORD IS $ ?       )
          5                 IF    QUESTION            ( - YES, ERROR        )
          6                 THEN                      (   ABORT             )
          7                 0 1 16                    (   HEX NUMBER        )
          8           ELSE $2D = DUPLICATE            ( - NO, - SIGN ?      )
          9                 BASE?                     (   CURRENT BASE      )
         10           THEN
         11
         12           NOTE PSW1 = BASE/$-BASE, 10/16                  *
         13                PSW2 = LEADING SKIP COUNT, 0/1 FOR $ AND -   *
         14                PSW3 = MINUS SIGN INDICATOR, 0/1 FOR -       *
         15                PSW4 = START ADDRESS OF WORD STRING          *. ->

SCR..8    0
          1           IBASE=                          ( SAVE BASE TO USE    )
          2           ROTATE +                        ( ADDRESS OF 1ST DIGIT)
          3           (NUMBER)                        ( CONVERT DIGIT STRING)
          4           C@ $20 -                        ( DIGIT STRING ENDED  )
          5                                           ( BY A SPACE CHAR. ?  )
          6           IF  QUESTION                    ( - NO, ERROR         )
          7               ( ABORTS )                  (       NOT A NUMBER  )
          8           THEN
          9           SWAP                            ( SIGN ADJUST FOR     )
         10           IF  NEGATE                      ( A NEGATIVE VALUE    )
         11           THEN
         12           ;
         13
         14
         15
```

*(Listing Continued)*

**PWS1010 8 Bit CPU Card (6801), 8K FORTH Firmware**
6K EEPROM, 2K RAM, 16 TTL I/O, RS 232C, programmable timer, 2K monitor, 8K FORTH firmware includes: editor, assembler, high-level interrupt linkage and communications protocol.

**PWS1080 16 Bit CPU Card (68008) 16K FORTH Firmware**
2K or 8K EEPROM, 8K RAM, RS 232C, 4 programmable timers. Non-multiplexed memory expansion and waitstate generator. 16K FORTH firmware includes: monitor, editor, assembler, high-level interrupt linkage and communications protocol.

**PWS2010 Interface Card, 8K extended FORTH firmware**
2 16-bit I/O ports, compatible with industry standard optical isolation boards, battery back-up calendar and clock, 2 28-pin JEDEC standard memory sockets for 2K or 8K CMOS, NMOS or EPROM memories, software readable 8-bit switch. Firmware options: P-FORTH Standard Utilities, P-FORTH PLC with Ladder Diagrams or P-FORTH Multi-Tasking.

**PWS2020 Interface Card 2 16-bit I/O ports**
Compatible with industry standard optical isolation boards.

# NEW FORTH BOARDS CARDS & STUFF

We're racing into tomorrow to give you a new family of control system products today. Our innovative FORTH team continues to introduce versatile, powerful and unique firmware with important advantages: low cost system development, interactive FORTH language to speed software creation, EEPROM nonvolatile memory and STD BUS interfacing. Look through our new FORTH firmware, we know you'll discover an application for your current or future projects. For samples, documentation or consultation call one of our FORTH team.

**PWS2030 Expansion Memory Board, JEDEC standard memory sockets**
Maximum of 64K EEPROM, EPROM or RAM.

**PWS3010 Color Video Graphics Card (TI9918)**
8K FORTH graphics firmware, 256x192 pixels, 15 color graphic RS 170 video output, 16K video RAM, ability to GENLOCK to external video, JEDEC Standard socket for additional firmware.

**PWS9010 STD BUS Card Cage**
6 Slots, mother board, integral power supply, 5 volts at 6 amps ± 12 volts at 1 amp, on/off and circuit breaker switch.

**CUSTOM MADE BOARDS**
We will custom make boards to your exact application if none of the aforementioned boards meet your needs.

**CONSULTING**
Our FORTH Team people are experts in FORTH based application development and are excited to be able to share their special knowledge with you on a personal basis.

*Watch for new products to come in 1983*

5190 West 76th Street
Minneapolis, MN 55435

**PEOPLEWARE SYSTEMS INC.**
(612) 831-0827 • TWX 910-576-1735

memory space. Four words are generated as follows

address of **VAR= ( A= )**
address of **VAR? ( A? )**
address of Word **CONSTANT ( A$ )**
address of data variable

The third and fourth words generated are the code for **VAR A$**. This discussion refers to a ROMable system. That is, the program code space is separated from the variable space. Hence the fourth word generated is an address, and not the contents as in most FORTH systems of the data variable.

The code for **VAR?** need only fetch the 16 bit word contents from a data address as given by a pointer which lies two words from where the code was called. The code is to end after popping the top word off the return stack. The code for **VAR=** is similar.

Secondly, the dictionary should be separated from the generated code in order to implement the code for **VAR⊏** and **VAR?**. The multiple dictionary entries can be implemented as follows:

- **IN** is saved; **IN** points to the position of the source program between **M-VAR** and **A.**
- **WORD** is used to scan the source Word **A** into a buffer area **HERE** in some FORTH systems); this buffer contains a one byte count (e.g. Word size of 3 for XYZ) followed by the characters of the name (e.g. **XYZ**).
- Adjust the buffer to the required name, e.g. add one to the count, insert the ASCII character = after the name.
- **CREATE** is used to create the dictionary; supply other fields (such as code address) as required for completing the dictionary entry.
- Restore **IN** and repeat the process two more times to obtain the names **A= A? A$.**

A listing of an implementation is given (see listing 2). Unfortunately, this will not work as is for a fig-FORTH system. No attempt has been made to supply a compatible listing as the implementation details of fig-FORTH are not familiar.

The listing applies to a system which

- scans all Words into a fixed buffer area rather than a changing **HERE.**
- the dictionary is automatically separated from program code (ROMable version) rather than preceding program code.

```
LISTING 2:    DATA STRUCTURE
                                                    VS2.0  JAN 83    PAGE- 1

SCR..1     0 NOTE      LISTING FOR DATA STRUCTURE
           1           -------------------------
           2
           3
           4           THIS SYSTEM ALREADY SEPARATED DICTIONARY
           5           FROM GENERATED CODE
           6
           7           CONSIDER
           8              1@  AS  @
           9              1!  AS  !
          10           BUT DEALS WITH THE DICTIONARY ( AND NOT CODE ) SPACE
          11
          12           FOR DEFINITIONS OF WORDS USED SEE
          13           CALTECH FORTH MANUAL - SECOND EDITION JUNE, 1978.
          14
          15           ->

SCR..2     0
           1
           2 NOTE      PSW1 = 1 FOR ITEM
           3                = 3 FOR ARRAY
           4           PSW2 = ? OR = OR $ OR L
           5
           6           ADJUSTS A DICTIONARY NAME
           7
           8           NOT A WORKING EXAMPLE - CONCEPTUAL IDEA ONLY
           9           ********************************************.
          10
          11           ->

SCR..3     0   : ADJUST-NAME
           1           CURRENT @ @ 4 +                  ( ADD. OF LENGTH FLD)
           2           DUPLICATE 1@                     ( LENGTH IN M.S.B.  )
           3           DUPLICATE 4 BRING 8 LSHIFT +     ( NEW LENGTH IN MSB )
           4           3 BRING 1!                       ( CHANGE LENGTH FLD )
           5           -8 LSHIFT + 1+                   ( NEXT NAME POSITION)
           6           SWAP 1-                          ( WHICH NAME TYPE?  )
           7           IF    SWAP $5B00 OR OVER 1!      ( - ARRAY,INSERT [X )
           8                 2+ $5D SWAP                (   AND SUPPLY ]    )
           9           THEN                             (   FOR INSERTION   )
          10           SWAP 8 LSHIFT SWAP 1!            ( INSERT X OR ]     )
          11           ;
          12
          13           ->

SCR..5     0 NOTE      [CREATE] IS THE ACTION FOR CREATE
           1           BUT CREATE IS IMMEDIATE
           2           WHILE [CREATE] IS NON IMMEDIATE
           3
           4           THE SAME APPLIES TO [CONSTANT]
           5                               [VAR]
           6                               [CVAR]
           7                               [ARRAY].
           8
           9
          10
          11
          12
          13
          14
          15           ->

SCR..6     0 NOTE      CODE ADDRESS IS NOT SUPPLIED
           1           SINCE THIS SYSTEM ALREADY
           2           SEPARATED DICTIONARY FROM GENERATED CODE.
           3
           4
           5
           6
           7
           8
           9   : CREATEX
          10           IN @ [CREATE] IN !               ( CREATE NAME )
          11           ADJUST-NAME
          12           ;
          13
          14
          15           ->

SCR..7     0 : CREATE-ITEM
           1           1 CREATEX
           2           ;
           3
           4
           5 : CREATE=
           6           $3D CREATE-ITEM          ( $3D IS HEX 3D FOR ASCII = )
           7           ;
           8 : CREATE?
           9           $3F CREATE-ITEM
          10           ;
          11
          12 : ADJUST-ADDRESS-ITEM-NAME$
          13           $24 1 ADJUST-NAME                ( GIVES $ )
          14           ;
          15           ->
```

*(Listing Continued)*

• interpretation is by calls to subroutines rather than threaded code. There is no code address generated for **COLON (:)** or **SEMI (;)**.
• variable/RAM memory space is automatically separated from program code (ROMable version) rather than intermixed with program code.

### Record Structure

Using the same technique as for data structure, a COBOL-like structure for contiguous memory space is easily implemented. Data can then be described as items or arrays of word, byte, bit(s), string or decimal. FORTH systems with program code space separated from the variable space (ROMable system) are particularly suitable for inclusion of such a record structure.

The record structure implemented consists of about 2K bytes of compiling code and 4K bytes of program code (ROMable code). This does not include dictionary space.

Typically citing **A?** or **A=** for a record structure involves an executive speed of between 4 to 6 times that of the Word **+**. The increase in execution speed due to runtime interpretation is found to be negligible for applications on Motorola M6800 and M6809 processors at IMHZ. A typical application program would be reasonably interrupt intensive, VDU display orientated and consist of over 64K bytes of ROMable code. Dictionary space, while heavily used, is not included in the target application.

### Implementation Details

A record data structure is described internally by
• the address of an executable environment pointer.
• the relative offset position from the start of the record structure.
• the internal encoded description of the data structure.

An environment pointer is allocated when a record header definition appears in the source program. All data structures within a record refer to the same environment pointer which may be switched by context or executed as required while the program is running.

The relative offset position is determined by the data structures

```
LISTING 2:    DATA STRUCTURE
                                                        VS2.0  JAN 83    PAGE- 2

SCR..8     0 : CREATE[X]
           1          CREATEX ;
           2 : CREATE-ARRAY
           3          3 CREATE[X]
           4          ;
           5 : CREATE[=]
           6          $3D CREATE-ARRAY
           7          ;
           8 : CREATE[?]
           9          $3F CREATE-ARRAY
          10          ;
          11 : CREATE[L]                                ( PSW1 = LIMIT      )
          12          IN @ OVER [CONSTANT] IN !         ( CREATE NAME       )
          13          $4C 3 ADJUST-NAME
          14          ;
          15          ->

SCR..9     0 NOTE     MAY NEED ONE MORE WORD
           1          DEPENDING ON WHETHER 'NEXT' INTREPRETATION IS
           2          PRE- OR POST- INCREMENT,
           3
           4          USE MACHINE CODE FOR SPEED
           5
           6          EXAMPLE FOR M6809 PROCESSOR
           7          AND JSR/RTS SUBROUTINE CALL IS
           8              PULS  X
           9              LDD   [3,X]          SUBROUTINE CALL IS 3 BYTES
          10              PSHU  D              PUSH INTO P-STACK
          11              RTS                  DOUBLE EXIT
          12          FOR DO-VAR?
          13          .
          14
          15          ->

SCR..10    0 NOTE     LEAVE OUT THE WORD @ ( NOT POINTED TO )
           1          FOR A NON ROMABLE SYSTEM
           2
           3          EG: R> 2+ @
           4              FOR DO-VAR?
           5          .
           6
           7          ->

SCR..11    0
           1
           2
           3
           4 : DO-VAR?
           5          R> 2+ @ @                      ( 2+ FOR 2 BYTES AWAY )
           6          ;                              (    EQUALS 1 WORD    )
           7
           8 : DO-CVAR?
           9          R> 2+ @ C@                     ( 4 + INSTEAD OF 2+   )
          10          ;                              ( FOR PRE-INCREMENT   )
          11                                         ( 'NEXT'              )
          12 : DO-ARRAY[?]
          13          R> 2+ @ + C@
          14          ;
          15          ->

SCR..12    0 : DO-VAR=
           1          R> 4 + @ !
           2          ;
           3
           4 : DO-CVAR=
           5          R> 4 + @ C!
           6          ;
           7
           8 : DO-ARRAY[=]
           9          R> 4 + @ + C!
          10          ;
          11
          12
          13
          14
          15          ->


SCR..13    0 NOTE     MODIFY M-VAR M-CVAR M-ARRAY AS REQUIRED
           1
           2          SEPARATE DICTIONARY FROM GENERATED CODE,
           3          SUPPLYING CODE ADDRESS AS REQUIRED
           4
           5          CHANGE ORDER OF ACTION
           6
           7          LISTING IS FOR SYSTEM
           8          WHICH ALREADY SEPARATED DICTIONARY FROM CODE
           9
          10          \ DO-VAR=
          11          IS EMPLACING WHICH GENERATES THE CODE.
          12
          13
          14
          15          ->
```

*(Listing Continued)*

following the record header definition, their order, and the size of each data item.

The internal description is an encoded internal representation of the data structure, and includes the number of elements for array data. During program execution, data structures are interpreted for action according to the internal description and checked for range.

An example of a record structure describing the FORTH dictionary is given (see listing 3). This is one of several different types of record structure. This particular record structure allows the description of the record to be switched to any entry to be cited by name. This is useful to describe linked lists, queues, tables, etc. Two examples of using this record structure are given.

### Example 1

To read the next dictionary entry use the statement

**LINK-ADDRESS? DICTIONARY-ENTRY$ READ-PICTURE-RECORD**

When **LINK-ADDRESS?** is cited the contents of the link field is placed on top of the stack. When **DICTIONARY-ENTRY$** is cited the address of the record data structure is placed on top of the stack. When **READ-PICTURE-RECORD** is cited the environment pointer is switched by context. Since the address of the environment pointer is the first word of the internal record data structure the code for the switch by context is simply

**: READ-PICTURE-RECORD @ ! ;**

Whenever **LINK-ADDRESS?** is now cited the contents of the new link field, and not the previous link field, will be accessed.

### Example 2

To invert the precedence (or immediate) bit, use the statement

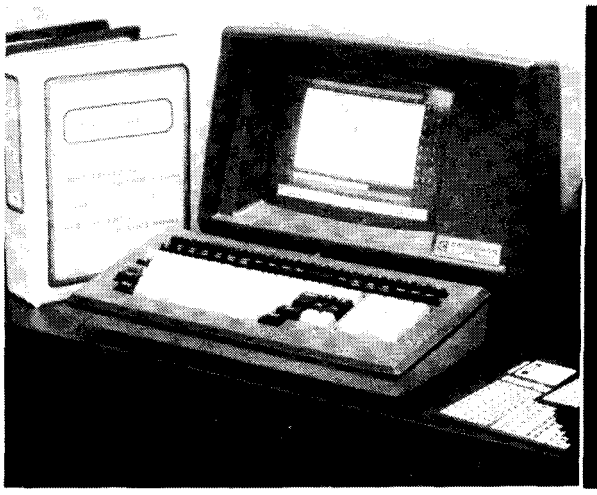
**PRECEDENCE-BIT? NOT PRECEDENCE-BIT =**

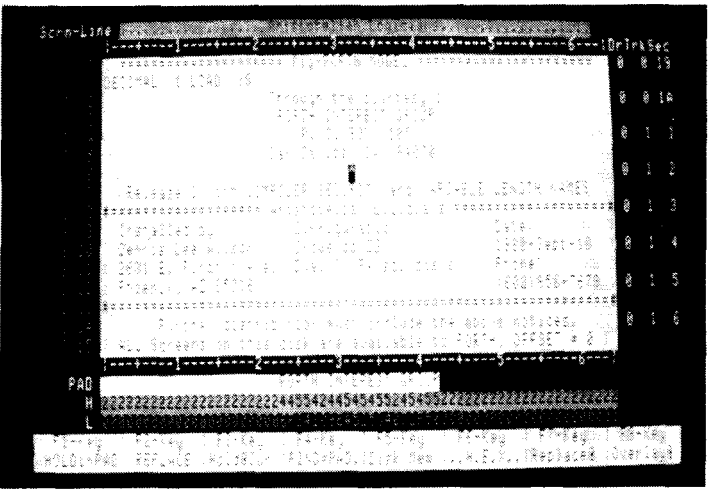
When **PRECEDENCE-BIT?** is cited, "bit 7" is placed on top of the stack at bit 0 with the high order 15 bits all zeros. This effect is referred to as transformation; that is, conversion of data types occurs automatically when data is cited by name. This can be simply illustated if **PRECEDENCE-BIT**

*(Continued on page 33)*

```
LISTING 2:    DATA STRUCTURE
                                                    VS2.0  JAN 83    PAGE- 3

SCR..15    0
           1
           2 : M-VAR                                          IMMEDIATE
           3          CREATE= \ DO-VAR=
           4          CREATE? \ DO-VAR?
           5          [VAR] ADJUST-ADDRESS-ITEM-NAME$
           6          ;
           7
           8 : M-CVAR                                         IMMEDIATE
           9          CREATE= \ DO-CVAR=
          10          CREATE? \ DO-CVAR?
          11          [CVAR] ADJUST-ADDRESS-ITEM-NAME$
          12          ;
          13
          14
          15          ->

SCR..16    0
           1
           2 : M-ARRAY                                        IMMEDIATE
           3          CREATE[L]
           4          CREATE[=] \ DO-ARRAY[=]
           5          CREATE[?] \ DO-ARRAY[?]
           6          [ARRAY] $24 3 ADJUST-NAME
           7          ;
           8


LISTING 3:    EXAMPLE OF RECORD STRUCTURE SHOWING FORTH DICTIONARY
                                                    VS2.0  JAN 83    PAGE- 1

SCR..1     0 NOTE     EXAMPLE OF RECORD STRUCTURE
           1
           2          RECORD DESCRIBES FORTH DICTIONARY
           3          MAXIMUM ENTRY SIZE IS SHOWN
           4
           5          THIS RECORD STRUCTURE DOES NOT ALLOCATE MEMORY SPACE.
           6
           7
           8
           9
          10
          11
          12
          13
          14
          15          ->

SCR..2     0
           1          ->
           2
           3
           4
           5          STRUCTURE                COBOL EQUIVALENT
           6          ---------                ----------------
           7
           8          M-1(16)                    PICTURE 1(16)
           9          M-1(5)                     PICTURE 1(5)
          10          M-1(1)                     PICTURE 1(1)
          11          M-FILLER-1(1)        FILLER PICTURE 1(1)
          12       31 M-1(8)-ARRAY               PICTURE 1(8) OCCURS 31
          13       31 M-X(.)                     PICTURE X(31)
          14
          15

SCR..3     0 M-PICTURE-RECORD          DICTIONARY-ENTRY     ( RECORD HEADER  )
           1
           2          M-1(16)            LINK-ADDRESS         ( 16 BIT FIELD )
           3          M-1(16)            CODE-ADDRESS
           4
           5          M-1(5)             LENGTH-OF-NAME       ( 5 BIT FIELD  )
           6          M-FILLER-1(1)                           ( SPARE BIT 5  )
           7          M-FILLER-1(1)                           ( SPARE BIT 6  )
           8          M-1(1)             PRECEDENCE-BIT       ( BIT 7        )
           9
          10       31 M-1(8)-ARRAY       NAME-ELEMENT         ( 31 BYTE ARRAY)
          11          M-REDO             NAME-ELEMENT[$]      ( REDEFINE NAME)
          12                                                  (    AS 31     )
          13       NAME-ELEMENT[L]                            (    CHARACTER )
          14          M-X(.)             NAME                 (    STRING    )
          15 M-END-RECORD                                    ( RECORD TRAILER )

SCR..4     0
           1
           2
           3
           4
           5
           6
           7
           8
           9
          10
          11
          12
          13
          14
          15
```

**End Listing**





# 8080/Z80 FIG-FORTH for CP/M & CDOS systems
# FULL-SCREEN EDITOR for DISK & MEMORY

$50 saves you keying the FIG FORTH model and many published FIG FORTH screens onto diskette and debugging them. *You receive TWO diskettes (see below for formats available). 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 and a special one for Cromemco 3102 terminals.

The 2nd disk contains FORTH readable screens including an extensive FULL-SCREEN EDITOR FOR DISK & MEMORY. This editor is a powerful FORTH software development tool featuring detailed terminal profile descriptions with full cursor function, full and partial LINE-HOLD LINE-REPLACE and LINE-OVERLAY functions plus line insert/delete, character insert/delete, HEX character display/update and drive-track-sector display. The EDITOR may also be used to VIEW AND MODIFY MEMORY (a feature not available on any other full screen editor we know of.) This disk also has formatted memory and I/O port dump *words and many items published in* FORTH DIMENSIONS, *including* a FORTH TRACE utility, a model data base handler, an 8080 ASSEMBLER and a recursive decompiler.

The disks are packaged in a ring binder along with a complete listing of the FULL-SCREEN EDITOR and a copy of 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 |
|---|---|---|
| ☐ **FIG-FORTH & Full Screen EDITOR package** | | |
| Minimum system requirements: | | |
| 80x24 video screen w/ cursor addressability | | |
| 8080 or Z80 or compatible cpu | | |
| CP/M or compatible operating system w/ 32K or more user RAM | | |
| Select disk format below, (soft sectored only) ..... | $50 | $65 |
| ☐ 8" SSSD for CP/M (Single Side, Single Density) | | |
| Cromemco CDOS formats, Single Side, S/D Density | | |
| ☐ 8" SSSD ☐ 8" SSDD ☐ 5¼" SSSD ☐ 5¼" SSDD | | |
| Cromemco CDOS formats, Double Side, S/D Density | | |
| ☐ 8" DSSD ☐ 8" DSDD ☐ 5¼" DSSD ☐ 5¼" DSDD | | |
| ***Other formats are being considered, tell us your needs.*** | | |
| ☐ Printed Z80 Assembly listing w/ xref (Zilog mnemonics) ..... | $15 | $18 |
| ☐ 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

# More on Data Bases

*Lindsay Doyle*

I learned a lot from Robert Watkins' article, "The Indexer" in Vol. IV, No. 5. However, I suspect that the degree of familiarity he assumed his readers would have with information-retrieval techniques may have made it difficult for newcomers to the field to grasp the concepts he was presenting. Also, readers should not assume that the bit-mapped approach he used is the only one or even necessarily the best one for solving his problem. On a third point he implies, it seems to me, that keywords can be added indiscriminately at any time to such a system, which is not the case. I would like to clarify these three points and perhaps a few others as well in the hope that the results may be helpful both to other readers and to Watkins in the further development of his system.

In the type of information retrieval system under discussion, records may consist of data held in computer storage or (and this is a significant point not mentioned in the referenced article) they may be physical documents such as books, catalogs, magazines, file folders, coins in a collection, etc. On acquisition (i.e., at the time it is entered into the file) each record is given a sequential record number and is characterized by selecting one or more keywords which describe its content or other aspects of significance to users. To retrieve such a record, the user selects keywords which he thinks will describe the record or type of record he is interested in and links these keywords with the logical operators **AND, OR, AND NOT**, and **OR NOT**. The system returns *all* records which fulfill the specifications. Other logical operators, not provided in Watkins' system, are also possible, such as <, >, **EARLIER THAN, LATER THAN**, etc.

The ideal situation is one where the person who designs the keyword list is also the only user of the system: where others use it, the question of personal interpretation and preferences on keywords may arise. In either case, it is important that the keywords used be taken from a previously constructed list which has been carefully inspected to make sure that it contains no synonyms or unintentional overlaps of meaning. If, when record number 1000 is being entered, a new keyword is invented to cover some apparently novel feature of that record, who is to say whether that feature exists, previously unrecognized, in the first 999 records? The only valid way to add a new keyword to the system is to review all existing records in the file to see whether it applies to them. This can also become a problem when the user's interests change. If I suddenly become interested in lasers, it does only a small amount of good to add the word "laser" to my list of keywords for indexing magazine articles, for only future articles will have that keyword applied unless I go back and review my entire library of potential laser-article-containing magazines and re-index them.

Watkins describes optical incidence or "peekaboo" cards where there is a card for each record and punch-hole positions on each card corresponding to all keywords, the positions corresponding to selected keywords being punched, the others not. The system he then implements is not the analog of this, which may have caused some readers to be confused. His system is the analog of an *inverted* file system, where there is a card for each *keyword* and punch-hole positions for each *record*. Such systems find limited use in their manual form because the number of punch-hole positions one can put on a reasonably-sized card is very small, thus limiting the number of records one can index. The computerized version has the advantage that the number of "punch-hole positions" (1's and 0's stored with the keyword) can be extended as far as one wishes. This system, which I shall refer to as the *bit-mapped*

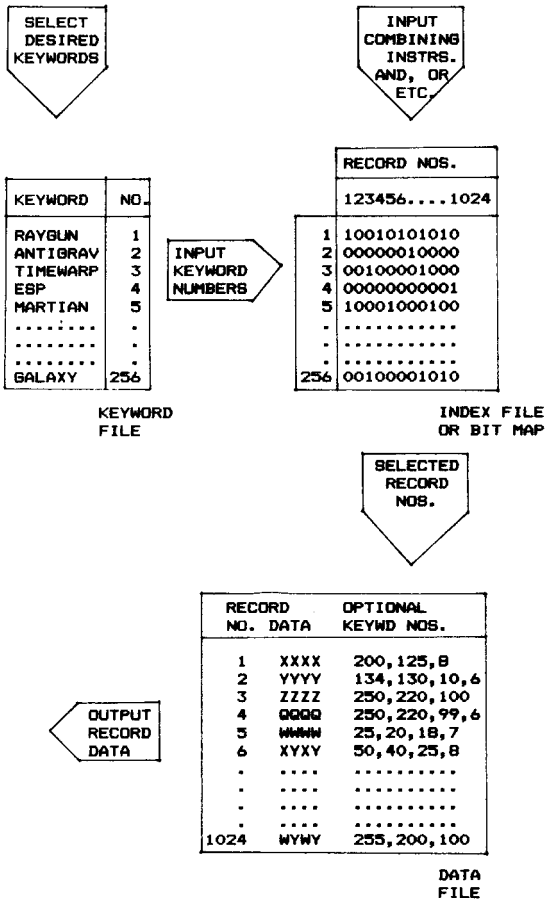



Fig. 1: Bit-Mapped Approach.

*approach*, is shown in Fig. 1. It requires three files to be maintained: the keyword file, the index file or bit map, and the data file. Watkins chose to include the keyword numbers for each record number within the record data, but as he points out, this is not required but is done to give a bit of redundancy in the event of damage to the bit map.

There is an improvement one can make both to the manual inverted card file system as described and to its computer equivalent. I don't know whether this has a name or not. It no longer allows the use of "peekaboo" detection in the manual case, but is very easy to implement in the computerized case. As before, there is a card for each keyword. Instead of the area of the card being mapped out in bits, one for each possible record, the selected record numbers are simply written on the card in increasing numerical order. Now the card area is being used much more efficiently. To say it another way, when a new record is to be added to the file, the desired keyword cards are removed from the card file and the new record number is added at the end of the list of record numbers on each card. This system, which I shall refer to as the *coded record number approach*, is shown in Fig. 2. It only requires two files and, as indicated by the broken line, the keyword file is open-ended.

Previously the maximum number of records that the entire file could hold was dictated by the number of dedicated punch-hole positions on any one card or the number of bits allowed per keyword in the computer version. Now the number of records the file can hold is no longer limited by the number which can be put on a card, but will be controlled by other system parameters. Clearly it is no longer possible to have a keyword which references every record, but such a keyword is useless in any case. The question now becomes: what percentage of the total records should one keyword be able to reference? My guess is that the answer is in the range of from 5% to 10%, and that keywords which reference more than this need to be retired or broken down into multiple, more-specific, new keywords.

Let us examine my statement above that "the card area is now being used more efficiently" as it applies to the computer version. Instead of allocating to each keyword one bit for every possible record, we must now encode the selected record numbers and record them with the keyword. The coding scheme used will define the maximum number of records the system can hold as well as the number of records which can be noted in a given-sized coding area against one keyword. If, for a first example, we limit ourselves to the same sized coding area that Watkins selected, which was 1024 bits or 128 bytes per keyword, some of the possible coding schemes are shown in Table 1.

The idea of being able to index 64K different records on a micro is very interesting. On the Commodore products, with which I do most of my work, there are 256 different symbols available counting graphics or their lower-case equivalents and reversed-video versions of everything. It is therefore possible to represent 16-bit numbers as two characters. A screen in the FORTH I use contains 1000 instead of 1024 characters, but let us stay with 1K which is more common. If we allow each keyword's record list to occupy one whole screen instead of only 128 bytes as in Table 1, what sort of performance can we get? Some possibilities are shown in Table 2.

To take this sort of investigation one step further, let us look at the case where the maximum number of records is the same for bit mapping and for two-byte coding. This occurs when 8192 bytes are allotted, and is shown in Table 3.

The penultimate cognitive jump to make is to realize that in any of these schemes the bit-map approach must have the total dedicated space assigned at all times. The coded versions, however, and taking case L as an example, may start with e.g. one screen per keyword and add extra screens by chaining as required, but only on those keywords which require it: i.e. it is not necessary to predict how many records per keyword one may end up with. One can tentatively conclude that the bit-map approach is suitable for small files which will not be called upon to expand beyond the

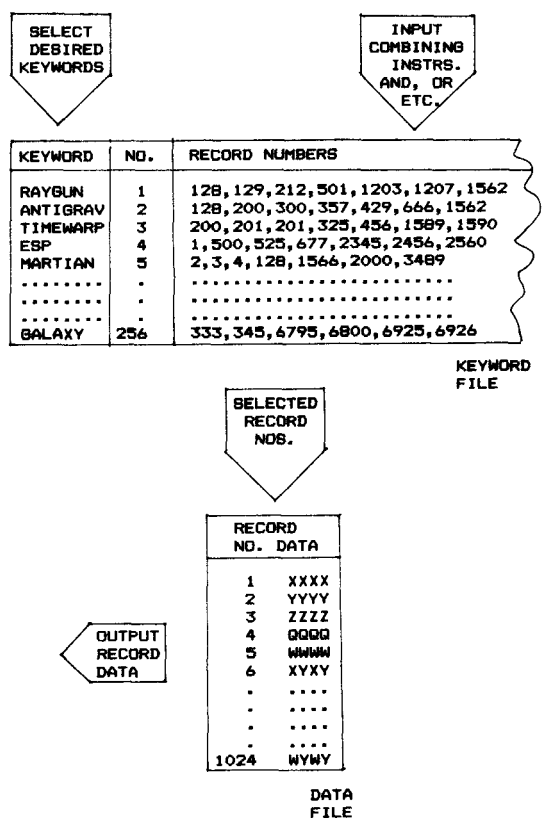



Fig. 2: Coded Record Number Approach.

originally-allowed-for limits, but that the coded approach has advantages where long files are concerned. I have not attempted to compare search techniques for the two, but would point out that the new words **>AND**, **>OR**, etc. developed by Watkins are not needed in the coded approach, as we are simply comparing pairs of 16-bit numbers, which can be done with the primitives >, =, and <.

My final contribution is the thought that in systems for indexing physical libraries, where we are dealing only with the record number and do not have to store the record data, the use of 16-bit numbers, expressed as two characters means not only that there is no longer a data file in the sense used by Watkins but also that a library containing up to 64K documents can be completely indexed for multiple-keyword retrieval on a few 5″ disks. Is anybody doing it?

*—Lindsay Doyle is a frequent contributor to* MicroComputer Printout *and other British microcomputer publications.*

| | BITS/ RECORD | RECORDS/ KEYWORD | MAXIMUM RECORDS | % OF MAX /KEYWORD | COMMENTS |
|---|---|---|---|---|---|
| A | 1 | 1024 | 1024 | 100 | BIT MAP |
| B | 8 | 128 | 255 | 50 | NOT AS GOOD |
| C | 12 | 85 | 4096 | 2 | HMM |
| D | 16 | 64 | 65535 | 0.1 | WOW! |

TABLE 1: 128 bytes per keyword or 8 keywords per screen.

| | BITS/ RECORD | RECORDS/ KEYWORD | MAXIMUM RECORDS | % OF MAX /KEYWORD | COMMENTS |
|---|---|---|---|---|---|
| E | 1 | 8192 | 8192 | 100 | BIT MAP |
| F | 8 | 1024 | 255 | >100 | POINTLESS |
| G | 12 | 682 | 4096 | 17 | INTERESTING |
| H | 16 | 512 | 65536 | 0.8 | WOW! |

Table 2: 1024 bytes per keyword or 1 keyword per screen.

| | BITS/ RECORD | RECORDS/ KEYWORD | MAXIMUM RECORDS | % OF MAX /KEYWORD | COMMENTS |
|---|---|---|---|---|---|
| I | 1 | 65536 | 65536 | 100 | BIT MAP |
| J | 8 | 8192 | 255 | >100 | POINTLESS |
| K | 12 | 5461 | 4096 | >100 | POINTLESS |
| L | 16 | 4096 | 65536 | 6+ | IN SUGGESTED RANGE |

Table 3: 8192 bytes per keyword or 8 screens.

# Paying the Piper

**The Problem:** Metal pipes cannot be manufactured without flaws. When exposed to high pressure and temperature, such flaws gradually deteriorate and eventually fail. Metal pipes are used in Nuclear Power Plants.

**The Solution:** Zetec Corporation devised a technique with which flaws in installed piping can be measured and analyzed, allowing for safe operation and life cycle maintenance of the large piping systems necessary in nuclear power plants.

The technique involves pushing a probe through the pipe. The probe emits vibrations which produce Eddy currents between the probe and the walls of the pipe. Flaws cause abnormalities in the frequency of the Eddy currents which are detectable by coils in the probe. This data is recorded onsite on an 8-channel analog data recorder and evaluated offsite by ZETEC personnel.

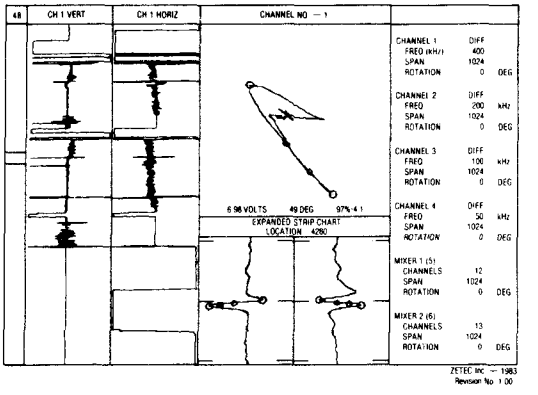
The Analog Data is read from the type, converted to digital, and analized by a Hewlett Packard Model 9836 68000 based computer running Multi-FORTH. The HP 9836 features built-in disc drives, an 80 x 24 alpha/512 x 390 graphics CRT, HPIB interface and up to 2¼ megabytes of RAM. Multi-FORTH provides a Real Time Multitasking programming environment. On the right is a graphics screen dump of a flaw. Note that the two columns on the left are a strip chart window into the much larger data base (in excess of 1 Mb of data). The current display shows a section of data recorded on Channel 1 at 400 kHz. The active channel and display mode are selected on the right side of the screen. Channel frequency, span of sample, and phasing rotation may be modified through a simple sequence of soft labeled special function keys.

The data in the small window in the far left most column is expanded in the lower center window, and the view size may be increased or decreased by the operator. The center upper window is an XY plot of the data in the expanded strip chart below. All windows are continuously refreshed in real time as the operator scrolls through the database. The "fuzzy balls" in both center windows are the result of a least squares fit on the data and further indicate the point of maximum deflection.

The results indicate 97% through wall flaw in the pipe at 49 degrees, at location 4280 in the pipe. Obviously this pipe is not in service.

Two mixer channels are provided to differentially remove the effects of structural supports for the pipe so that flaws under such supports may be detected.

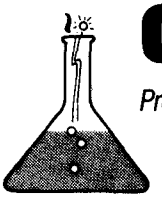
Under Multi-FORTH the strip chart windows, tape control and analog to digital conversion, and analytical functions operate as multiple background tasks, while the operator interface operates in the foreground, i.e., the strip charts may be scrolling up or down through the database while the operator is selecting channels, viewing data, or initiating computations.




We think that Lloyd Lamb and Howard Houserman of ZETEC deserve a lot of credit. After attempting this application with HP Basic (an unusually fast Basic at that), they stopped after three months due to insufficient display performance. The project was converted and completed in Multi-FORTH in about 1½ months. While both men are acknowledged industry experts in Eddy current technology and data acquisition, neither has had extensive training in computer science.

Gentlemen, we salute you. (And boy are we proud!)

MULTI-FORTH IS AVAILABLE ON ALL HEWLETT PACKARD SERIES 200 COMPUTERS (9816, 9826, 9836) AND MOST 68000 SINGLE BOARD COMPUTERS: KDM, VM01, VM02, OB68K-1, BRI, EXORMACS, VME110, ENG696, DUAL. FOR MORE INFORMATION GET IN TOUCH WITH US.



**Creative Solutions Inc.**

*Problem Solving For Business and Computer Applications*

4801 Randolph Road Rockville, MD 20852

Phone (301) 984-0262

ANOTHER CSI CUSTOMER APPLICATION SUMMARY

## New Product Announcements

### METACRAFTS FORTH FOR APPLE II/ IIe

Metacrafts Limited announces the release of Metacrafts FORTH V1.2 for Apple II/ IIe computers with 48K RAM and at least one Disk II drive.

The system, which runs the Byte prime number benchmark in 164 seconds, includes: 79 Standard required word set; double number extension; CASE; strings; arrays; on-stack local variables; vocabulary stacks; dictionary overlays; heap store; block buffer control; hi- and lores graphics (including turtle graphics); I/0 execution vectors; macro assembler; full-screen editor with "undo" and "syntax-check" features; interactive source-level debugger; intelligent multi-block copy; memory dumper; threaded code decoder; paginated printer output; screen INDEX; and OUTLINE words. Support for 40/80 column display; language card; and multiple disk drives. 100+ screens of source code for upper levels of system. 170-page User's Guide (assumes knowledge of FORTH). Price: £79,00 (+15% VAT in UK) includes shipping. Manual only, £8,00. Information leaflet available. Dealer inquiries welcome. Contact: *Metacrafts Lts., 144 Crewe Rd., Shavington, Crewe CW2 5AJ, England. Phone: (0270) 666274.*

### FORTH COMPUTER

This C-MOS Eurocard module gives faster software and hardware development times than assembler level programming. Software costs in industrial applications cannot be amortized over the large quantities associated with personal computers and electronic games. This C-MOS embedded computer card aims at resolving this problem by including FORTH high level language programming and developmental facilities. Using FORTH rather than machine assembler gives a fast reaction time to market opportunities. Production products use the same board as employed in the prototypes.

No microprocessor development system is needed since the card contains a screen editor working with simple visual display units (VDUs). It also has the compiler for the FORTH source code. Debugging is inherent in the FORTH language and once the code is working, this can be output to a PROM programmer.

Use of C-MOS throughout has brought the power consumption down to 28mA, making the TDS900 especially suitable for portable and battery-driven applications. The TDS900 price is £179.95 and in hundreds versions are available at £87 - £120. In the U.S. the FORTH Computer is available from *Stynetic Systems Inc., Flowerfield, Bldg. 1, Saint James, NY 11780 (516) 862-7670.*

### Extensible Text Formatter/Editor

QTF+ is a powerful documentation tool, ideal for technical writing and documentation of FORTH programs. The formatter features string and formatting macro defining capability; justification; centering; tabbing; hanging indent; automatic page and chapter numbering; page headers and footers; soft hyphenation; automatic page number referencing to tables, figures and topics elsewhere in your document; and formatting of FORTH screens, as well as full utilization of the Epson printer fonts, underlining and boldface. The cursor-controlled wrap-around text editor offers insert, delete, replace, and string-move. Authored by Leo Brodie. Requires IBM PC with 64K running Laboratory Microsystems' PC/FORTH. Available for $50.00 from: *Laboratory Microsystems, 4147 Beethoven St., Los Angeles, CA 90066.*

### LOOK-SEE

The SOFT-WRIGHTS' new screen design package FORTH LOOK-SEE provides a model for the design of screen/menu input for FIG 8080 FORTH. LOOK-SEE allows the user to design screen/menu input in a manner that is similar to the way the screen/menu is to look to the user. This provides extreme ease in the design/maintenance/update of screen/menu driven programs in FORTH, thus reducing costs and design change turnaround time.

Cost: $10.00 postpaid in the U.S. Comes as a listing with a clear concise user manual. LOOK-SEE utilizes user screen I/0. Contact: *THE SOFT-WRIGHTS, 840 Van Ness #107, San Francisco, CA 94109*

### FORTH-79 VER. 2 FOR Z-80 CP/M & APPLE USERS

MicroMotion has announced an expanded line of formats available for Z-80 1.4 & 2.x users. These include APPLE, Micropolis Mod II, Vector Grahics, Micropolis & Tandon, North-Star, Cromemco, Heath/Zenith, Osborne I, Kaypro II, Xerox 820, and TRS-80 Model II. Meets all provisions of the FORTH-79 Standard. Base system includes a screen editor, macro-assembler, string-package, 3-bit integer arithmetic and 200 page tutorial and reference manual. Floating Point available for all versions, HIRES for APPLE & Northstar. $99.95-$139.95. Contact: *MicroMotion, 12077 Wilshire Blvd., #506, Los Angeles, CA 90025, 213/821-4340.*

### SOFTWARE WORKSHOPS IN MMSFORTH

Miller Microcomputer Services introducers a regular series of Boston-area MMSFORTH Workshops on a variety of topics and ability levels.

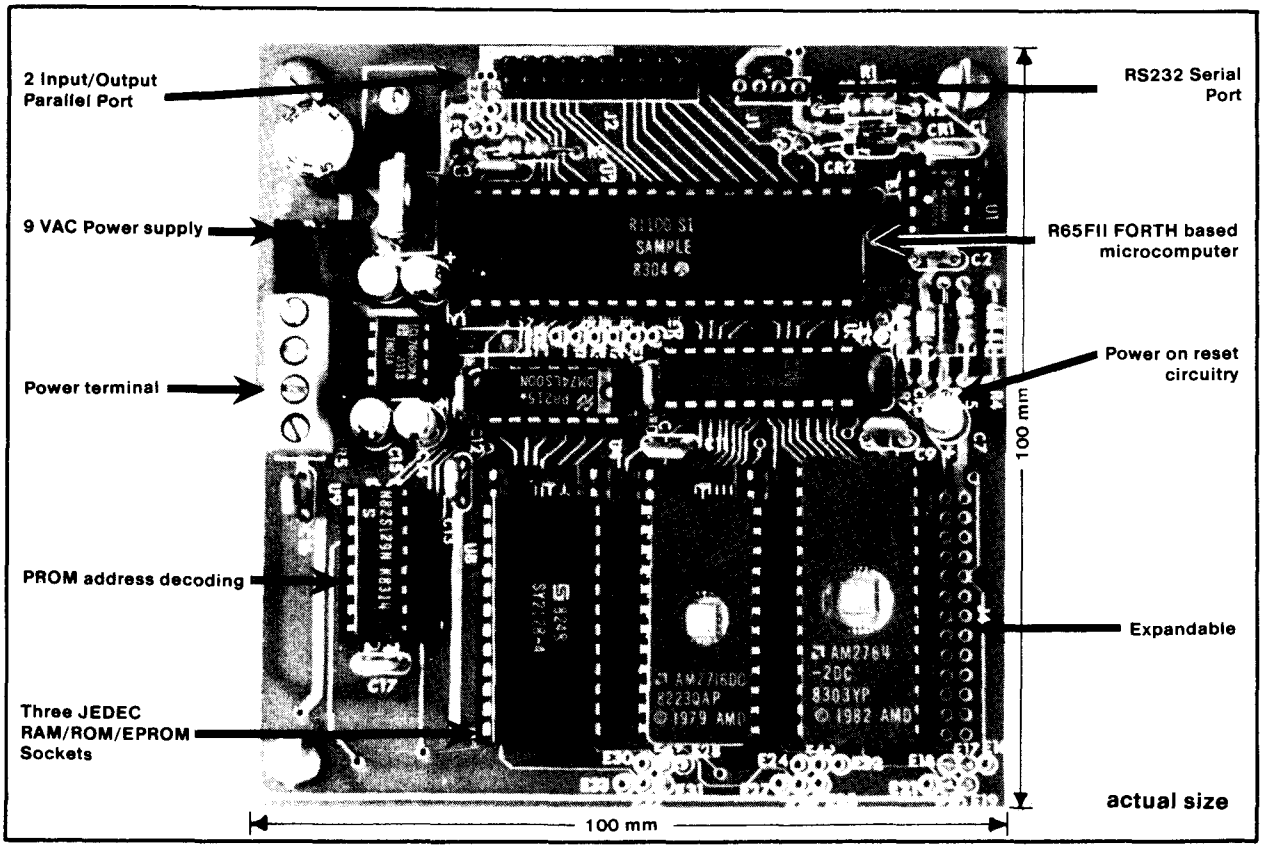
The schedule of offerings for the remainder of the 1983 is as follows.

| Workshop | Level | Cost | Days |
|---|---|---|---|
| Introduction to MMSFORTH | Elem | $250 | 2 |
| Applications in MMSFORTH | Int | $250 | 2 |
| Database Design and Implementation | Adv | $950 | 5 |
| MetaForth & Other Advanced Topics | Adv | $950 | 5 |
| Forthwrite User Techniques | Elem | $150 | 1 |
| Datahandler User Techniques | Elem | $150 | 1 |
| Forthcom User Techniques | Elem | $150 | 1 |

Schedules and further information are available from Miller Microcomputer Services, 61 Lake Shore Road, Natick, MA 01760 (617/653-6136, 9 a.m. to 6 p.m. Eastern Time Zone).

# A PREMIER OFFERING TO THE FORTH COMMUNITY!

## *A limited number of R65F11 Microcomputer FORTH Development System - at a special price . . .*




By the time you read this ad we should receive our first shipment of production R65F11 Microcomputers, the 6502 based single chip microcomputers with the run time portions of FORTH in ROM. This chip features a complete FORTH based operating system and is ideal for dedicated microcomputer applications. Our board, the NMIX-0011, surrounds the R65F11 with equally innovative circuitry that allows the chip to be a complete FORTH development system. (We call the board the "100 squared" for short, due to its extremely small size). All that is needed to do program development in FORTH is a CRT terminal or microcomputer that speaks RS232 (seven data, one start, two stop bits).

**Look for a complete Euro card boardline coming soon -**

The "100 squared" features on board rectification and regulation of power from a 9 volt AC or DC power source. Terminals are there if you prefer to use your own regulated 5V supply. An on board DC to DC convertor can provide negative voltage for the RS232 interface either way. Address decoding is accomplished by a bi-polar PROM that can be replaced by the user if necessary. A standard development PROM decoder is provided with the board. Three JEDEC 28 pin sockets are provided which will accept:

| | |
|---|---|
| *RAM's* | *2016, 2128, 5517, 6116, 5564* |
| *EPROM's* | *2716, 2732, 2764* |
| *EEPROM's* | *2816A* |

The board can program in circuit:

| | |
|---|---|
| *R2816A* | *2764** |

**requires additional VPP voltage supply.*

All this plus the powerful R65F11 which features:

- *FORTH kernel in ROM*
- *Enhanced 6502 CPU*
- *192-byte static RAM*
- *16 bidirectional, TTL-compatible I/O lines (two ports, R65F11)*
- *One 8-bit port with programmable latched input*
- *Two 16-bit programmable counter/ timers, with latches*
- *Serial port*
- *Ten interrupts*
- *Expandable to 16K bytes of external memory*
- *Flexible clock circuitry*
- *1 us minimum instruction execution time @ 2 MHz*
- *NMOS silicon gate, depletion load technology*
- *Single* +5V power supply
- 12 mW standby power for 32 bytes of the 192-byte RAM
- 40-pin DIP (R65F11)

We will be advertising very soon in the major trade journals. We anticipate demand to be so great that this will quickly become a limited availability item. We wanted to offer it first to the people that made the R65F11 possible - the people involved with the FORTH Interest Group. We are offering a special order price of $220.00. This is $30 off our list price, but, to reserve your board WE MUST HAVE YOU ORDER **NOW!** This is a limited time offering! ACT NOW.

***Enclose Payment With Order To:***

New Micros, Inc.
2100 N. Hwy. 360
Suite 1607
Grand Prairie, Texas 75050
(214) 660-1106
Telex 79-5551

was described as decimal with the structure **M-9(1)** which corresponds to **PICTURE 9(1)** in COBOL. In this case the data variable is either ASCII 0 or ASCII 1 in memory while **PRECEDENCE-BIT?** when cited places binary 0 or binary 1 on top of the stack. When **NOT** (or **0=**) is cited bit 0 of the top word of the stack is inverted; the high order 15 bits are irrelevant.

When **PRECEDENCE-BIT=** is cited bit 0 of the top word of the stack is stored in 'bit 7'. The **LENGTH-OF-NAME** field is not manipulated even though **PRECEDENCE-BIT** lies in the same memory byte.

To do the same in conventional FORTH the following is required.

```
FETCH-ADDRESS-OF-LINK-FIELD
4 + DUP C@ $80 XOR SWAP C!
```

The following irrelevant code exists

- the offset is required to be known (**4**).
- the address of the precedence bit is required to be specifically calculated (**+**).
- the structure of the memory cell is required to be known to be a byte (**C@ C!**).
- the position of the precedence bit is required to be known to be in bit 7 (**$80**).
- the memory address is manipulated on the stack (**DUP SWAP**).

Furthermore, the code is not self-documenting; it is not obvious that the precedence bit is inverted.

**Conclusion**

It is desirable to extend the FORTH compiler. Experience shows it is viable to adapt features from another language to FORTH. This is best done in FORTH source to allow portability and could be implemented by runtime interpretation for an increase in execution speed.

Many areas of the FORTH compiler need addressing to incorporate language/operating system features. With the availability of large development systems for generating target applications, falling memory prices, and languages with an increasing number of features, the time may not be too far away when very large compilers are readily available. As to memory size for the FORTH compiler, why not heed the directive to "go forth and multiply"?



## FORTH Vendors *(Continued from page 43)*

Metalogic Corp.
4325 Miraleste Dr.
Rancho Palos Verdes, CA 90274
213/519-7013

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

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

Schleisiek, Klaus
Eppendorfer Landstr. 16
D 2000 Hamburg 20
West Germany
(040)480 8154

Schrenk, Dr. Walter
Postfach 904
7500 Karlstruhe-41
West Germany

Software Engineering
6308 Troost Ave. #210
Kansas City, MO 64131
816/363-1024

Softweaver
P.O. Box 7200
Santa Cruz, CA 95061
408/425-8700

Technology Management, Inc.
1520 S. Lyon St.
Santa Ana, CA 92705
714/835-9512

Timin, Mitchel
3050 Rue d'Orlean #307
San Diego, CA 92110
619/222-4185

## Technotes

### ENCLOSE Encounters

*Nicholas L. Pappas, Ph.D.*

Here are two tests and a practical solution for the **ENCLOSE** bug.

In many applications a natural decision to take is

**HEX 1 ' B/SCR !** (blocks/screen = 1)
**400 ' B/BUF !** (bytes/block buffer = 1024)

One consequence is the enclose bug may get you.* The ENCLOSE primative in many Forth listings uses a one byte counter to accumulate the (blank) delimiter and character counts. When B/BUF holds less than 100 hex, the counter's capacity is adequate. however, a two byte counter is needed when B/BUF holds a number greater than the one byte counter's FF capacity.

If a screen has more than FF delimiters in a row, call this a bad string; two kinds of wrong events may occur. A bad string before the end of the screen will cause the system to lock up so that OK is not returned. We have a crash requiring a reset. A bad string elsewhere may confound loading the screen (see IN in WORD) or crash. Testing for existence of the enclose bug is easy to do. Fill screen in with blanks, set b/scr = 1, b/buf = 1024, and n LOAD. If you get OK, all is okay. A more direct test is to set up for 1024 byte buffers and

**HEX FIRST 2 + 400 BLANKS**
(blanks a block buffer)

If the response prints as 400 401 400 addr, there is no bug. If the response prints 0 1 0 addr, you have the bug.

A simple, practical solution is breaking up bad strings with a harmless ( )**, and a long term solution is to rewrite* your ENCLOSE primitive using a two byte counter.

**for a gotcha and an 8080 rewrite, see* Forth Dimensions, *Vol III, #2, pages 35 and 41.*
***2 blanks or more.*

*(Continued)*

## Technotes *(Continued)*

### U/ Bug Fumigated

*K.G. Lander, Crewe, England*

In "Technotes" IV-D Vol. IV, pg. 2, Jack Haller identified a bug in **U/**, (79 standard **UMOD**) for which he proposed a correction. Unfortunately his solution is only partially correct, as the following entries demonstrate:

1048576. 65535 U/MODok
U. 31 ok (Quotient should be 16)
U. 31 ok (Remainder should be 16)

The problem is that the **N 1+ ROL,** instruction on line 7 accumulates ALL the carries that would have been lost by the original version, instead of just the latest. The corrected version (see listing) clears N+1 after the remembered 17th bit has been processed.

Incidentally, while checking out the corrected solution, I discovered that the **U*** carry bug correction in the Installation Model has not been incorporated into the 6502 Assembly listing.

### A Better CRC

*Mike Steckmyer*

I am writing to comment about your article entitled: "Checksum for Hand-Entered Source Screens" which appeared in FORTH IV, 3. I became interested in the algorithm after reading the part about being patient. Waiting up to 2 seconds per screen was noted. This seems to be a little too long.

One of my past projects was to develop a Pseudo-Random Binary Sequence (PRBS) register. PRBS is just another name for a CRC register. As you suggested, an evaluation of the given algorithm was in order.

I simulated the algorithm to determine if it has a maximal length sequence. A PRBS register will cycle through states and eventually start repeating. The number of states before repeating is the length of the register. A 16 bit register is maximal if it has a 2*16 state PRBS. I found that the register in "ACCUMULATE" is 2*15 states long.

The number 4002 H (16386) defines the feedback used in the given algorithm. I found that a feedback value of 148C H (5260) produced a maximal length PRBS. I also determined that the byte shift left (256 *) and bit shift left loop (8 0 DO ... LOOP) did not effect the length of the sequence. So both operations can be removed without a loss of performance and help improve the speed of checksumming.

I propose using this new definition:

**: PRBS** (oldcrc/char — newcrc)
**XOR DUP 0< IF 5260 XOR DUP + (SHL) 1 + ELSE DUP + THEN ;**

I found that "PRBS" executes 9 times faster than "ACCUMULATE" because there is far less computing being done. It should also be noted that this optimization produces a different PRBS than the original algorithm.

## Loaded in Decimal Mode

```
SCREEN #99
 0 ( CORRECTED U/MOD
 1 CODE U/MOD
 2       N 1+ STY,
 3       SEC 2+ LDA,                 SEC LDY,
 4       SEC 2+ STY,     ,A ASL,     SEC STA,
 5       SEC 3 + LDA,                SEC 1+ LDY,
 6       SEC 3 + STY,    ,A ROL,     SEC 1+ STA,
 7       16 # LDA, N STA,
 8       BEGIN, SEC 2+ ROL,  SEC 3 + ROL, N 1+ ROL,
 9            SEC, SEC 2+ LDA,  BOT SBC,  TAY,
10            SEC 3 + LDA,  BOT 1+ SBC,  PHA,
11            N 1+ LDA, 0 # SBC, 0 # LDA, N 1+ STA, PLA,
12            CS IF,  SEC 2+ STY,  SEC 3 + STA,  THEN,
13            SEC ROL, SEC 1+ ROL,  N DEC, 0= UNTIL,
14       POP JMP,
15 END-CODE
```


# FORTH
# $150

## DESIGNED BY EXPERTS IN THE FIELD OF MICRO-COMPUTERS
## RICHARD ALTWASSER AND STEVEN VICKERS

**Steven Vickers**
Steven gained his degree in Math at King's College, Cambridge, England, and his Ph.D in Algebra at Leeds University. His first assignment after school was to create the Sinclair ZX-81 or Timex 1000 8K ROM, and to write the ZX-81 manual. Subsequently he wrote most of the ROM for the Sinclair Spectrum or Timex 2000.

**Richard Altwasser**
Richard gained his honors degree in Engineering at Trinity College, Cambridge, England. He joined Sinclair in September 1980, and was instrumental in the research that led to the development of the Spectrum or Timex 2000.

Recently these two experts started their own company and developed the Jupiter Ace range of hardware which is based on the exciting new language for micro-computers "FORTH".

**For the FORTH enthusiast**
The Jupiter Ace closely follows the FORTH 79 standard with extensions for floating point, sound and cassette. It has a unique and remarkable editor that allows you to list and alter words that have been previously compiled into the dictionary. This avoids the need to store screens of source, allowing the dictionary itself to be saved on cassette. Comprehensive error checking removes the worry of accidentally crashing your programs.

Order Form:
Send To
Computer Distribution Assoc.
56 South 3rd Street
Oxford, Penna. 19363

| Product | Price | Qty. | Total |
|---|---|---|---|
| Jupiter Ace | $150 | | |
| 16K Ram Pack | $ 50 | | |
| 48K Ram Pack | $125 | | |
| Par/Ser Interface | $100 | | |
| Shipping and Handling | | $4.95 | |
| Total Order | | | |

Credit Card No. Exp. Date

Signature

FD IV/6

# A Simple Overlay System

*Christian Mahr*

This article presents a simple overlay system. The overlays are binary images of pre-compiled FORTH code, i.e. programming tools or other ready-to-run programs.

In my FORTH system I like to have all the programming tools at hand that I need in program generation and testing: the assembler, editor, disk-maintenance-programs, disassembler, the FORTH-discompiler and some others. Most of them are mutually exclusive; they refer only to definitions of the main FORTH system but not to definitions of each other. Unfortunately, these tools are eating up a lot of valuable dictionary space, decreasing the amount of free memory for my application programs. On the other hand, a loading (compiling) of a tool for each time of use would be very time consuming. Some tools can't be discarded from memory after use because they generate code themselves, i.e. the assembler.

These facts in mind, I wrote this overlay system to rapidly load a tool as a binary image. All tools use the same memory space, counted backwards from the end of available dictionary memory. So there is only need for enough memory for the tool I actually want to use. The overlay system is divided into a resident part located behind the system definitions, and a second part that is loaded during the generation of a binary overlay image only.

The following concept is certainly a BFBI approach (brute force and bloody ignorance), but it works well for my needs: An overlay occupies a multiple of 1K Byte counted backwards from the end of available dictionary memory (determined by my system constant EM). The first eight words of an overlay block contains the following loading and linking information:

block address +
  0 contains the start
  block of the overlay

```
 0 ( overlay system - resident section                cm  05.03.83 )    80
 1   FORTH DEFINITIONS HEX                                              80
 2 9FFF CONSTANT EM  DECIMAL       ( end of availiable dic memory )     80
 3   0  VARIABLE OV#                 ( current overlay number )         80
 4   0  VARIABLE OVA                 ( current overlay addr   )         80
 5 : X ;    -4 ALLOT            ( the input  dummy link definition )    80
 6 : Y ;    -4 ALLOT            ( the output dummy link definition )    80
 7 : RELINK   [ ' Y LFA ] LITERAL !  ;       ( link into overlay )       80
 8 : CANCEL   ' X  NFA  RELINK       0 OV# !  ;  ( cancel overlay )      80
 9                                                                      80
10 : GET.IMAGE                      ( addr\start-blk\dr#\blks ... )     80
11       ( load <blks> contingous blocks from disk, starting at )       80
12       ( <start-blk>, drivr <dr#>, destination address <addr> )       80
13      8 *  >R ( sectors )  SWAP 8 * >R   (  start sector )            80
14           >R ( drive # )  1 ( do a read )   R> R> R> -DISK ;         80
15 DECIMAL -->                                                          80

 0 ( overlay system - resident section                cm             )   81
 1 : OVERLAY   ( blk ... )              ( create an overlay caller )    81
 2     <BUILDS   DUP OFFSET @ + 250 /MOD SWAP , ,                       81
 3      DUP BLOCK   DUP @ ROT -  12 ?ERROR       ( valid overlay? )     81
 4      4 + HERE  4 CMOVE  4 ALLOT          ( copy load information )   81
 5     DOES>   DUP 6 + @  HERE U< 2  ?ERROR        ( does not fit )      81
 6       DUP @  OV# @ -  IF  ( not in memory )  CANCEL                  81
 7       >R  R 6 + @   R       @                                        81
 8           R 2+  @   R  4 + @   GET.IMAGE  ( load into memory )       81
 9       R 6 + @   ( addr )                                             81
10       DUP @   R> @ -   12  ?ERROR          ( wrong block )          81
11       DUP   8 + @                      ( this is LFA of first def )  81
12       [ ' X NFA ] LITERAL  SWAP !       ( link X to the first )      81
13       DUP  10 + @  RELINK            ( this is NFA of last def )     81
14       DUP OVA !  @ OV# !                    ( set OVA and OV# )      81
15     ELSE  DROP  THEN  ;    -->                                       81

 0 ( overlay system - resident section                cm             )   82
 1 : EXG  ( addr1 addr2 ... )          ( exchange variable contents )    82
 2       DUP >R @ SWAP DUP @ R> ! !  ;                                  82
 3                                                                      82
 4 : OV-SWITCH  ( ... ) ( switch from main dictionary to overlay )      82
 5     CURRENT @ OVA @ 10 + EXG                  ( area and back )      82
 6     DP        OVA @ 12 + EXG                                         82
 7     VOC-LINK  OVA @ 14 + EXG  ;                                      82
 8                                                                      82
 9 HEX  A0   ' X NFA 1+  C!                      ( kill name of X )     82
10     A0   ' Y NFA 1+  C!                      ( kill name of Y )     82
11                                                                      82
12 DECIMAL  ;S       ( end of resident section of overlay system )      82
13                                                                      82
14 don't forget to place this error message on scr# 4 line 12:         82
15  " no valid overlay block found ! "                                  82

 0 ( overlay system - non-resident section          cm  05.03.83 )     83
 1 ( These definitions are necessary to create the binary images )      83
 2 ( and save them to disk. Please see if your system has         )     83
 3 ( 1024 Bytes/block and 250 Blocks/drive or not. )                    83
 4                                                                      83
 5   FORTH DEFINITIONS DECIMAL                                          83
 6                                                                      83
 7 : SAVE.IMAGE   ( addr \ start-blk \ dr# \ blks ... )                 83
 8       CR ." writing overlay image on screens "                       83
 9       >R 250 * +   R>   OVER + SWAP DO                               83
10         DUP  I . I  OFFSET @ -  BLOCK   B/BUF CMOVE  UPDATE          83
11       B/BUF +   LOOP  DROP  FLUSH ;                                  83
12 -->                                                                  83
13                                                                      83
14                                                                      83
15                                                                      83


 0 ( overlay system - non-resident section          cm             )    84
 1                                                                      84
 2 : UNLINK  ( ...link1\link2 ) ( unlink the overlay definitions )      84
 3                                              ( from main dic )       84
 4     LATEST    CR ."  unlink: "                                       84
 5     BEGIN                                                          84
 6       DUP ID. 2 SPACES                  ( print unlinked )           84
 7       DUP  PFA LFA @   OVA @ U< 0= WHILE                             84
 8       PFA LFA @                                                      84
 9     REPEAT  CR    ;                                                84
10 -->                                                                  84
```

*(Listing Continued)*

```
 0 ( overlay system - non-resident section          cm          )
 1
 2 : OPEN-OVERLAY ( abs.blk, blks ... )      ( open up space for )
 3              ( creation of an overlay image with size <blks>  )
 4              ( starting on <abs.blk> ; memory aligned to EM   )
 5       CANCEL                ( cancel a possible overlay before ! )
 6       DUP B/BUF * MINUS EM 160 - +          ( overlay adress )
 7       DUP  HERE U< 2 ?ERROR                 ( enough room ?  )
 8       DUP  OVA !                          ( is overlay adress )
 9       HERE  OVER 12 + !  DP !           ( save DP, new DP=OVA )
10       SWAP  250 /MOD SWAP , , ,        ( strt-blk, dr#, blks, )
11       OVA @ ,                            ( save load adress )
12       0 ,                                  ( save out-link )
13       LATEST  , 2 ALLOT                    ( save in-link  )
14       VOC-LINK @  ,        ;               ( save VOC-LINK )
15 -->

 0 ( overlay system - non-resident section          cm          )
 1
 2 : CLOSE-OVERLAY  ( ... )  ( close overlay area )
 3       UNLINK   PFA LFA  OVA @  8 + !          ( save out-link )
 4       OV-SWITCH
 5       EM HERE U< 13 ?ERROR           ( overlay area too small )
 6       OVA @ >R
 7       R 6 + @   R @  R 2 + @  R> 4 + @  SAVE.IMAGE
 8       CANCEL  ;
 9 ;S
10
11 don't forget to place this error message on scr# 4 line 13:
12    " overlay area too small !"
13
14
15

 0 ( overlay system - non-resident section          cm  23.12.82 )
 1 ;S
 2
 3 PFA of an overlay caller:
 4   BLK  DR#    BLKS  OVA
 5   +0    +2    +4    +6
 6
 7 parameter-block of the overlay:
 8   BLK  (DR#)  BLKS  OVA   OUT-LINK  IN-LINK  OV-DP  VOC.SAVE
 9   +0    +2    +4    +6    +8        +10      +12    +14
10
11
12
13
14
15

 0 ( load screen for overlay assembler )
 1  FORTH DEFINITIONS                 : TASK ;       ( marker )
 2
 3  83 LOAD     ( the non-resident overlay generator definitions )
 4
 5  232 6 OPEN-OVERLAY  ( overlay begins at scr# 232, reserve    )
 6                      ( 6 kByte = 6 screens                    )
 7  50 LOAD             ( load the source text of the ASSEMBLER  )
 8                      ( to be compiled into the overlay area   )
 9
10 ( see how far the overlay area is filled: )
11 CR  ." last memory location =$ "  HERE HEX 0 D. DECIMAL
12
13  CLOSE-OVERLAY   ( reset pointers to main dictionary and save )
14  FORGET  TASK    ( the binary image to disk, forget all       )
15 ;S

 0 ( load screen for overlay assembler )
 1
 2 232 OVERLAY ASSOV          ( create the overlay caller ASSOV, )
 3                            ( overlay is on screen 232         )
 4
 5   ASSOV     ( get this overlay for the following definitions )
 6
 7 ( make the following definition global by including the      )
 8 ( overlay caller and redefine this definitions :            )
 9
10 : ASSEMBLER    ASSOV  [COMPILE] ASSEMBLER  ;
11 : CODE         ASSOV  [COMPILE] CODE       ;
12 : M:           ASSOV  [COMPILE] M:         ;
13 : LABEL        ASSOV  [COMPILE] LABEL      ;
14 : EQUATE       ASSOV  [COMPILE] EQUATE     ;
15 ;S
```

*(Listing Continued)*

2 is reserved
4 contains the number of blocks to be loaded

6 contains the overlay start address

8 contains the output link address

10 contains the input link address

12 contains the overlay dictionary pointer

14 contains the overlay VOC-LINK

To invoke (load from disk if necessary) an overlay caller must be defined in the main dictionary. The PFA of this caller looks similar to the first 4 bytes in the overlay:

PFA of the overlay-caller +
0 contains the start block of the overlay
2 contains the drive # to load the overlay from
4 contains the number of blocks to be loaded
6 contains the overlay start address

If you now want to convert a program or programming tool to an overlay, proceed as follows:

1. Make the resident section of the overlay system a part of your cold-start system, or make sure that it is always loaded to the same address (scr# 80 to 82).
2. Your program becoming an overlay should be debugged; calculate the size of it in K byte and determine some continuous free blocks on the disk where the binary image will reside. If you proceed as shown on screen #88, the image will be created and saved on disk. No dictionary space will be consumed after this procedure, and this compilation is only done once.
3. To invoke the overlay, you have to compile what I have named an overlay caller into the main dictionary, as shown on scr #89, where the assembler overlay caller ASSOV is created. Each time ASSOV is executed, it checks whether this overlay is in memory and loads it from disk if not. Now you can make some key definitions "global", by redefining them with the overlay caller included. In the example, the

word CODE will automatically ask for the right overlay.

In order to not waste time while loading a binary image, I highly recommend you use a fast multiple block loader like the one which is accessible in my FORTH. In case you don't need or want it, use the high level version on scr #91.

There is an extra goodie when keeping the assembler in a separate overlay area: the use of local labels. It is possible to compile local **LABEL**s or **EQUATE**s as constants into the overlay area which is never written back to disk. These labels are valid as long as the assembler overlay is not canceled, but they consume no memory of the main dictionary. A proposal for such a LABEL and EQUATE definition is shown on scr #90.

In conclusion, I want to mention some points that I regard as drawbacks or as open questions:

- How to handle multiple vocabularies from the inside to the outside of an overlay. This system requires one link into and one out of the overlay. A vocabulary inside the overlay has to begin and end within this overlay.
- If anything in the FORTH system up to the resident part of the overlay system is changed (recompilation of the system) the images have to be recompiled as well.
- How to relocate a precompiled image. Perhaps by using an additional link table (?).
- How to invoke an overlay from another and return to the old after execution.
- If the compiler flags an error during generation of an overlay, keep in mind that the DP is now in the overlay area while compiling: First FORGET everything before trying to compile the overlay again.

If you have difficulties in adapting this program to your system: I mainly use a FIG-FORTH version on a 6809 or a 6502 with 1024 Byte/Buffers and 250 Blocks/Drive.

```
 0 ( 6809 assembler - label                          cm  23.12.82 )    90
 1 ( equate and label uses the overlay )                               90
 2                                                                     90
 3 FORTH DEFINITIONS                                                   90
 4                                                                     90
 5 : EQUATE    ( create a temporary constant in the overlay area )     90
 6           ?EXEC  [COMPILE] ASSEMBLER   OV-SWITCH                    90
 7           CONSTANT                                                  90
 8           LATEST RELINK  OV-SWITCH    ; IMMEDIATE                   90
 9                                                                     90
10 : LABEL       ( create a temporary label in the overlay area )     90
11           HERE   [COMPILE]  EQUATE ;    IMMEDIATE                   90
12                                                                     90
13 ASSEMBLER DEFINITIONS     ;S                                       90
14                                                                     90
15                                                                     90
```

```
 0 ( overlay system - resident section               cm  05.03.83 )    91
 1                                                                     91
 2 ( alternate definition for GET.IMAGE if you have no direct )        91
 3 ( access to loader for fast multiple block loading )                91
 4                                                                     91
 5                                                                     91
 6 : GET.IMAGE                    ( addr\start-blk\dr#\blks ... )     91
 7         ( load <blks> contingous blocks from disk, starting at)     91
 8         ( <start-blk>, drive <dr#>, destination address <addr>)     91
 9         SWAP 250 * OFFSET !                                         91
10         OVER + SWAP DO                                              91
11            I BLOCK OVER B/BUF CMOVE                                 91
12            B/BUF +                                                  91
13         LOOP  DROP ;                                                91
14 DECIMAL   ;S                                                        91
15                                                                     91
```

**End Listing**

## Glossary

addr = 16-bit address
n = signed 16-bit integer
d = signed 32-bit integer

### resident section

**EM** constant gives end address of available dictionary memory in your system.
**OV#** variable keeps the screen number of the current overlay in memory, zero means no valid overlay in memory.
**OVA** variable keeps the address where the current overlay begins.
**X** and **Y** are dummy definitions that are renamed to blank and used to link the overlay into the main dictionary.
**RELINK** nfa ... links this nfa into the dummy definition Y.
**CANCEL** no stack effect cancels the current overlay.
**GET.IMAGE** addr start.blk dr# blks ... loads <blks> continuous blocks from disk, drive <dr#>, starting at block <start.blk> to the destinatikon address <addr>. Accesses the disk driver directly, not via BLOCK.
**EXG** addr1 addr2 ... exchanges the contents of addr1 and addr2.
**OV-SWITCH** no stack effect switches from overlay area to main dictionary and back.

### non-resident section

**SAVE.IMAGE** addr start.blk dr# blks ... writes <blks> continuous blocks to disk, drive <dr#>, starting at block <start.blk> from the source address <addr>.
**UNLINK** no stack effect unlinks all definitions in overlay area from the main dictionary.
**OPEN-OVERLAY** start.blk blks ... sets pointers for compilation into the overlay area; the overlay shall start at <start.blk> and be <blks> KByte of size.
**CLOSE-OVERLAY** no stack effect stops compilation into overlay area, unlinks overlay from main dictionary, saves it to disk and resets pointers back to main dictionary.

# FIG Chapter News

### Southern Ontario

The fifth quarterly meeting of Fig Southern Ontario was held on March 5, 1983 with 15 people present. During the meeting, Dr. N. Solntseff outlined the FORTH-related work being carried out within the Unit for Computer Science at McMaster University.

**Implementation**

*a) Continuing work on implementation of fig-FORTH 78/79* on Ohio Scientific Instruments C2-8P and C3 computers. The aim of this work is to investigate floppy-disc, Winchester, and OS interfaces, as well as achieving improvements to the fig model.

*b) Development of a portable FORTH system* (undergraduate project to be completed by April 1, 1983). The idea behind this is the design of pseudo-microcode to implement an Abstract FORTH Machine (see paper with this title in 1982 Rochester Conference Proceedings, pp. 157-160). The FORTH-system kernel has been rewritten in the pseudo-microcode (called Fcode, by analogy with the Pcode of the Pascal system). Currently this implementation strategy is being tested via an implementation on McMaster's CYBER/170 with a microcode interpreter written in ETH Pascal. A macroprocessor for generating host-computer machine code is being designed.

*c) Implementation of figFORTH78 on a VAX11/780 under VMS* (graduate project). This project is complete and is being written up.

**Applications**

*a) Animated-Graphics Tutorial System* (funded by McMaster's) Instructional Development Centre—pilot project completed). A script-based tutorial development system has been designed capable of acting as an "animated text-book" to be used in conjunction with computer organization and computer architecture classes. The present version is too machine dependent for general distribution as it is based on OSI equipment.

*b) High-Level FORTH Graphics Primitives* (undergraduate project—April 1, 1983 completion). This project is aimed at the comparison of several line-drawing and other plotting primitives that could be used to provide a machine-independent graphics system.

*c) Graphics Primitives for the TRS-80 Colour Computer* (undergraduate project—April 1, 1983 completion). This project complements (b) above and is looking at graphics primitives for the TRS-80 Colour Computer.

*d) Document Preparation System* (graduate project—December 1983 completion). The aim of this project is to combine graphics with word processing to produce a system capable of handling lecture notes for computer organization and computer architecture courses. (An IBM PC is being used as the development system.)

# Fig Chapters

## U.S.

### • ARIZONA

**Phoenix Chapter**
Dennis L. Wilson
Samaritan Health Services
2121 E. Magnolia
Phoenix, AZ
602/257-6875

### • CALIFORNIA

**Los Angeles Chapter**
Monthly, 4th Sat., 11 a.m.
Allstate Savings
8800 So. Sepulveda Boulevard
Los Angeles
Phillip 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, 4th Wed., 12 noon.
Fullerton Savings
18020 Brookhurst
Fountain Valley
714/523-4202

**San Diego Chapter**
Weekly, Thurs., 12 noon.
Call Guy Kelly
714/268-3100 ext.4784

### • 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.
MNFIG
1156 Lincoln Avenue
St. Paul, MN 55105
Call Mark Abbot (days)
612/854-8776 or
Fred Olson
612/588-9532

### • MISSOURI

**St. Louis Chapter**
Call David Doudna
314/867-4482

### • NEVADA

**Las Vegas Chapter**
Suite 900
101 Convention Center Drive
Las Vegas, NV 89109
702/737-5670

### • NEW JERSEY

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

### • NEW YORK

**New York Chapter**
Call Tom Jung
212/746-4602

### • OKLAHOMA

**Tulsa Chapter**
Monthly, 3rd Tues., 7:30 p.m.
The Computer Store
4343 South 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 Granger
513/849-1483

### • OREGON

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

### • PENNSYLVANIA

**Philadelphia Chapter**
Continental Data Systems
1 Bala Plaza, Suite 212
Bala Cynwid, PA 91004
Call Barry Greebel

### • 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

**Vermont Fig Chapter**
Monthly, 3rd Mon., 7:30 p.m.
Vergennes Union High School
Room 210, Monkton Road
Vergennes, VT 05491
Contact Hal Clark
RD #1 Box 810
Starksboro, VT 05487
802/877-2911 days;
802/453-4442 eves.

### • VIRGINIA

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

## FOREIGN

### • AUSTRALIA

**Australia Chapter**
Contact Lance Collins
65 Martin Road
Glen Iris, Victoria 3146
(03)292600

### • CANADA

**Southern Ontario Chapter**
Contact Dr. N. Solntseff
Unit for Computer Science
McMaster University
Hamilton, Ontario L8S 4K1
416/525-9140 ext. 2065

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

### • ENGLAND

**English Chapter**
FORTH Interest Group
38 Worsley Road
Frimley, Camberley
Surrey, GU16 5AU, England

### • JAPAN

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

### • NETHERLANDS

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

### • WEST GERMANY

**West German Chapter**
Klaus Schleisiek
FIG Deutschland
Postfach 202264
D 2000 Hamburg 20
West Germany

## SPECIAL GROUPS

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

**Detroit Atari FORTH**
Monthly, 1st Wed.
Call Tom Chrapkiewicz
313/524-2100 or
313/772-8291

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

**MMSFORTH Users Groups**
Monthly, 3rd Wed., 7 p.m.
Cochituate, MA
Dick Miller
617/653-6136
(25 groups world-wide)

# FORTH System Vendors

*(by Category)*

**(Codes refer to alphabetical listing e.g., A1 signifies AB Computers, etc.)**

### Processors

| | |
|---|---|
| 1802 | C1, C2, F3, F6, L3 |
| 6502 (AIM, KIM, SYM) | R1, R2, S1 |
| 6800 | C2, F3, F5, K1, L3, M6, T1 |
| 6801 | P4 |
| 6809 | C2, F3, L3, M6, S11, T1 |
| 68000 | C2, C4, D1, E1, K1 |
| 68008 | P4 |
| 8080/85 | A5, C1, C2, F4, I5, L1, L3, M3, M6, R1, T3 |
| Z80/89 | A3, A5, C2, F4, I3, L1, M2, M3, M5, N1, T3 |
| Z80000 | I3 |
| 8086/88 | C2, F2, F3, L1, L3, M6 |
| 9900 | E2, L3 |

### Operating Systems

| | |
|---|---|
| CP/M | A3, A5, C2, F3, I3, L3, M1, M2, M6, T3 |
| CP/M86 | C2 |

### Computers

| | |
|---|---|
| Alpha Micro | P3, S3 |
| Apple | A4, F4, I2, I4, J1, L4, M2, M6, M8, 02, 03 |
| Atari | M6, P2, Q1, V1 |
| Compaq | M5 |
| Cromemco | A5, M2, M6 |
| DEC PDP/LSI-11 | C2, F3, L2, S3 |
| Heath-89 | M2, M6, M7 |
| Hewlett-Packard 85 | |
| Hewlett-Packard 9826/36 | C4 |
| IBM PC | A8, C2, F3, L1, M5, M6, Q2, S9, W2 |
| IBM Other | L3, W1 |
| Kaypro II/Xerox 820 | M2 |
| Micropolis | A2, M2, S2 |
| North Star | I5, M2, P1, S7 |
| Nova | C5 |
| Ohio Scientific | A6, B1, C3, O1, S6, T2 |
| Osborne | M2 |
| Pet SWTPC | A1, A6, B1, C3, O1, S6, T2, T5 |
| Poly Morphic Systems | A7 |
| TRS-80 I, II, and/or III | I5, M2, M5, M6, S4, S5, S10 |
| TRS-80 Color | A3, A8, F5, M4, S11, T1 |
| Vector Graphics | M2 |

### Other Products/Services

| | |
|---|---|
| Applications | P4 |
| Boards, Machine | F3, M3, P4, R2 |
| Consultation | C2, C4, N1, P4, T3, W1 |
| Cross Compilers | C2, F3, I3, M6, N1, P4 |
| Products, Various | A5, C2, F3, I5, S8, W2 |
| Training | C2, F3, I3, P4, W1 |

# FORTH Vendors *(Alphabetical)*

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
4. Applied Analytics Inc.
   8910 Brookridge Dr., #300
   Upper Marlboro, MD 20870
5. Aristotelian Logicians
   2631 E. Pinchot Ave.
   Phoenix, AZ 85016
7. Abstract Systems, etc.
   RFD Lower Prospect Hill
   Chester, MA 01011
8. Armadillo Int'l Software
   P.O. Box 7661
   Austin, TX 78712
   512/459-7325

**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
   England KT16 9LA
3. Consumer Computers
   8907 La Mesa Blvd.
   La Mesa, CA 92041
   714/698-8088
4. Creative Solutions, Inc.
   4801 Randolph Rd.
   Rockville, MD 20852
5. Capstone Computing, Inc.
   5640 Southwyck Blvd., #2E
   Toledo, OH 43614
   419/866-5503

**E**

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

**F**

1. Fantasia Systems, Inc.
   1059 The Alameda
   Belmont, CA 94002
   415/593-5700
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
   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 Ave.
   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.
   Suite #202B
   Portland, OR 97219
   503/244-4181

5. Interactive Computer Systems, Inc.
6403 Di Marco Rd.
Tampa, FL 33614

**J**

1. JPS Microsystems, Inc.
361 Steelcase Rd., W.
Markham, Ontario
Canada L3R 3V8
416/475-2383

**K**

1. Kukulies, Christoph
Ing. Buro Datentec
Heinrichsallee 35
Aachen, 5100
West Germany

**L**

1. Laboratory Microsystems
4147 Beethoven St.
Los Angeles, CA 90066
213/306-7412
2. Laboratory Software Systems, Inc.
3634 Mandeville Canyon
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 St.
Jersey City, NJ 07302
201/451-2905

**M**

1. M & B Design
820 Sweetbay Dr.
Sunnyvale, CA 94086
2. MicroMotion
12077 Wilshire Blvd., #506
Los Angeles, CA 90025
213/821-4340
3. Microsystems, Inc.
2500 E. Foothill Blvd., #102
Pasadena, CA 91107
213/577-1477
4. Micro Works, The
P.O. Box 1110
Del Mar, CA 92014
714/942-2400
5. Miller Microcomputer
61 Lake Shore Rd.
Natick, MA 01760
617/653-6136
6. Mountain View Press
P.O. Box 4656
Mountain View, CA 94040
415/961-4103
7. MCA
8 Newfield Ln.
Newtown, CT 06470
8. Metacrafts Ltd.
Beech Trees, 144 Crewe Rd.
Shavington, Crewe CW1
5AJ
England

**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" St.
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 Mgmt. Services
724 Arastradero Rd., #109
Palo Alto, CA 94306
408/252-2218
4. Peopleware Systems Inc.
5190 West 76th St.
Minneapolis, MN 55435
612/831-0872

**Q**

1. Quality Software
6660 Reseda Blvd., #105
Reseda, CA 91335
2. Quest Research, Inc.
P.O. Box 2553
Huntsville, AL 35804
800/558-8088

**R**

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 Hts., IL 60004
312/259-1355
7. Software Works, The
1032 Elwell Ct., #210
Palo Alto, CA 94303
415/960-1800
8. Supersoft Associates
P.O. Box 1628
Champaign, IL 61820
217/359-2112
9. Satellite Software Systems
288 West Center
Orem, UT 84057
801/224-8554
10. Spectrum Data Systems
5667 Phelps Luck Dr.
Columbia, MD 21045
301/992-5635
11. Stearns, Hoyt Electronics
4131 E. Cannon Dr.
Phoenix, AZ 85028
602/996-1717

**T**

1. Talbot Microsystems
1927 Curtis Ave.
Redondo Beach, CA 90278
2. Technical Products Co.
P.O. Box 12983
Gainsville, FL 32604
904/372-8439
3. Timin Engineering Co.
C/o Martian Technologies
8348 Center Dr. Suite F
La Mesa, CA 92041
619/464-2924
4. Transportable Software
P.O. Box 1049
Hightstown, NJ 08520
609/448-4175

**V**

1. Valpar International
3801 E. 34th St.
Tucson, AZ 85713
800/528-7070

**W**

1. Ward Systems Group
8013 Meadowview Dr.
Frederick, MD 21701
2. Worldwide Software
2555 Buena Vista Ave.
Berkeley, CA 94708
415/644-2850

**Z**

1. Zimmer, Tom
292 Falcato Dr.
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 Dr., #200
Portland, OR 97211
503/284-8277

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

Triangle Digital Services Ltd.
23 Campus Road
London E17 5PG
England

**Application Packages Only**
See System Vendor Chart for others

Curry Associates
P.O. Box 11324
Palo Alto, CA 94306
415/322-1463

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

**Consultation & Training Only**
See System Vendor Chart for others

Bartholomew, Alan
2210 Wilshire Blvd. #289
Santa Monica, CA 90403
213/394-0796

Boulton, Dave
581 Oakridge Dr.
Redwood City, CA 94062

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

Eastgate Systems Inc.
P.O. Box 1307
Cambridge, MA 02238

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

Intersystems Management Computer Consultancy
Story Hill Rd. RFD3
Dunbarton, NH 03045
603/774-7762

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

McIntosh, Norman
2908 California Ave., #3
San Francisco, CA 94115
415/563-1246

*(Continued on page 33)*

## FORTH INTEREST GROUP

## MAIL ORDER

| | USA | FOREIGN AIR |
|---|---|---|
| ☐ Membership in FORTH Interest Group and Volume V of FORTH DIMENSIONS | $15 | $27 |
| ☐ Back Volumes of FORTH DIMENSIONS. **Price per each.** ☐ I ☐ II ☐ III ☐ IV | $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 Listings of fig-FORTH for specific CPU's and machines. The above manual is required for installation. Check appropriate box(es). **Price per each.** ☐ 1802 ☐ 6502 ☐ 6800 ☐ 6809 ☐ VAX ☐ Z80 ☐ 8080 ☐ 8086/8088 ☐ 9900 ☐ APPLE II ☐ ECLIPSE ☐ PACE ☐ NOVA ☐ PDP-11 ☐ 68000 ☐ ALPHA MICRO | $15 | $18 |
| ☐ "Starting FORTH" by Brodie. BEST book on FORTH. (Paperback) | $18 | $22 |
| ☐ "Starting FORTH" by Brodie. (Hard Cover) | $22 | $27 |
| ☐ PROCEEDINGS 1980 FORML (FORTH Modification Lab) Conference | $25 | $35 |
| ☐ PROCEEDINGS 1981 FORML Conference, Both Volumes | $40 | $55 |
| ☐ Volume I, Language Structure | $25 | $35 |
| ☐ Volume II, Systems and Applications | $25 | $35 |
| ☐ PROCEEDINGS 1982 FORML Conference | $25 | $35 |
| ☐ PROCEEDINGS 1981 FORTH Univ. of Rochester Conference | $25 | $35 |
| ☐ PROCEEDINGS 1982 FORTH Univ. of Rochester Conference | $25 | $35 |
| ☐ FORTH-79 Standard, a publication of the FORTH Standards Team | $15 | $18 |
| ☐ Kitt Peak Primer, by Stevens. An in-depth 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, August 1980 BYTE cover, 16" x 22" | $ 3 | $ 5 |
| ☐ FORTH Programmer's Reference Card. If ordered separately, send a stamped, addressed envelope. | | FREE |
| ☐ Dr. Dobb's Journal, Two FORTH Issues, 9/81 & 9/82 | $ 7 | $10 |
| **TOTAL** | $_____ | |

NAME_____ MAIL STOP/APT_____

ORGANIZATION_____ PHONE (  )_____

ADDRESS_____

CITY_____ STATE_____ ZIP_____ COUNTRY_____

VISA #_____ MASTERCARD #_____

Expiration Date_____ (Minimum of $15.00 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. 1/83

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

Address Correction Requested