<table>
<thead>
<tr>
<th>Page</th>
<th>Title</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>137</td>
<td>Functional Programming and Forth</td>
<td>Harvey Glass</td>
</tr>
<tr>
<td>138</td>
<td>Forth and Artificial Linguistics</td>
<td>Raymond Weisling</td>
</tr>
<tr>
<td>140</td>
<td>Technotes</td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>A Forth Assembler for The 6502</td>
<td>William F. Ragsdale</td>
</tr>
<tr>
<td>151</td>
<td>A Technical Tutorial: Table Lookup Examples</td>
<td>Henry Laxen</td>
</tr>
<tr>
<td>152</td>
<td>The Game of Reverse</td>
<td>M. Burton</td>
</tr>
<tr>
<td>154</td>
<td>The 31 Game</td>
<td>Tony Lewis</td>
</tr>
<tr>
<td>156</td>
<td>Simulated Tektronics 4010 Graphics with Forth</td>
<td>Timothy Huang</td>
</tr>
<tr>
<td>158</td>
<td>A Video Version of Master Mind</td>
<td>David Butler</td>
</tr>
<tr>
<td>162</td>
<td>Transfer of Forth Screens by Modem</td>
<td>Guy T. Grotke</td>
</tr>
</tbody>
</table>
EDITOR'S COLUMN

1981 is behind us and as I look back, I am pleased to see how much has been accomplished for FORTH, FIG and FORTH DIMENSIONS.

I really appreciate all the help and support I have received from our readers. I have not done everything right and some of the best help has been the disagreement. Intelligent, constructive criticism is as welcome as earned praise.

1982 will be a year of continued growth. You can look forward to continuing responsiveness. It is my plan to contact every FIG chapter by telephone at least quarterly to get feedback and encourage reader contributions.

FORTH DIMENSIONS will also be awarding AUTHOR'S CERTIFICATES for outstanding articles that contribute to the growth and understanding of the language. While we are not yet in a position to give you cash for your contributions, we at least will give you credit.

Starting in this issue will be a policy of putting in tutorial articles designed to help our entry level readers. This, however, will not be done at the expense of our more seasoned FIGGERS who will find an expanded base of challenging articles and applications.

In closing, I want to say that the writer's kits have finally come off the presses and I will be glad to send one to anyone who wants to contribute. Please send in applications and utilities, philosophy, questions and problems -- in the final analysis, FORTH DIMENSIONS is what you make it.

C. J. Street

PUBLISHER'S COLUMN

1981 has been a great year for FORTH, the FORTH Interest Group and for me, personally. FORTH has spread around the world and is being used on thousands of computer and microprocessor-based products. It is being taught extensively in schools, companies and by FORTH programmers. FIG has just completed its most successful national convention with almost 500 attendees, over 20 exhibitors and multiple sessions. (Thanks to Bob Reiling, Conference Chairman and Gary Feterbach, Program Chairman.) The FORML conference was well attended and the Proceedings are now available--see order form.

My deepest thanks to the FORTH community for "THE FIGGY", Man of Year Award. It was a fantastic thrill and a surprise. I stand in good company.

Roy C. Martens

HISTORICAL PERSPECTIVE

FORTH was created by Mr. Charles H. Moore in 1969 at the National Radio Astronomy Observatory, Charlottesville, VA. It was created out of dissatisfaction with available programming tools, especially for observatory automation.

Mr. Moore and several associates formed FORTH, Inc. in 1973 for the purpose of licensing and support of the FORTH Operating System and Programming Language, and to supply application programming to meet customers' unique requirements.

The Forth Interest Group is centered in Northern California. Our membership is over 2,400 worldwide. It was formed in 1978 by FORTH programmers to encourage use of the language by the interchange of ideas through seminars and publications.
LETTERS TO THE EDITORS

Dear Fig:

I have developed a process-simulation program that occupies very little memory space and yet has many of the capabilities of commercial simulation packages.

I have been heavily involved in modeling and simulation of automated manufacturing systems for over six years. My ultimate objective for this work is to develop a microprocessor-based simulation capability which incorporates process control structures far beyond those of currently available languages. However, the relatively extensive modeling power of the current code would seem to offer interesting market potential in its own right.

If you can provide information on marketing such a product, please contact me by mail or by phone (home (317) 447-9206, office (317) 749-2946).

Joseph Talavage, Ph.D.
3907 Prange Dr.
Lafayette, IN 47905

Hope printing your letter helps.—ed.

Dear Fig:

I am puzzled as to why I have not seen mention in your New Products announcements of fullFORTH+ for PET, available also, I believe for Apple. It is published by IDPC, Co., PO Box 11594, Bethlehem Pike, Colmar, PA 18915 at $65. It is advertised as "A full-featured FORTH with extensions conforming to Forth Interest Group standards. Includes assembler, string processing capabilities, disk virtual memory, multiple dimensioned arrays, floating point and integer processing." Surely, fullFORTH+ is worth a mention, if not a comprehensive review!

Francis T. Chambers
ROCK HOUSE
Ballyhour, Westport
Co. Mayo, Ireland

Thank you for your interest. This was reviewed in Vol. III, #3.—ed.

Dear Fig:

This is our response to Chuck's (Moore's) cute letter.

Arthur Goldberg
Spencer Soo-Hoo
CEDARS SNAI MEDICAL CTR.
9700 Beverly Blvd.
Los Angeles, CA 90048

DEA- CHU--

WE CHA------ YOU TO CON------ THE
CON------ OF THE CON------ LET------. WE
CLA-- THA-- THE CON------ OF A WOR-- IN
COM------ ENG------. CON------ CON------
- TO OUR ABI-- TO DE------ IT FRO--
THR-- LET------ AND THE LEN------.

HOW------, IN COM------ PRO------ THE
CON------ IS LES-- CON------ (COM------,
WOR-- CAN BE SWA------ WIT------ CHA------
- THE SEM------). CON------ IT CON------
----- CON------ LES-- HEL-- IN IDE------
--- THE DEF------ OF A WOR------ IN FAC--
, AS THI-- LET------ DEM------. THI--
LET------ AND A LEN------ CAN LET------
EVE-- A CAR------ REA-- OF COM------
ENG------.

SIN------ YOU--,
ART-- GOL------
SPE-- SOO------

(TRANSLATION)

Dear Chuck:

WE CHALLENGE YOU TO CONSIDER
THE CONTENTS OF THIS CONFUSING
LETTER. WE CLAIM THAT THE
CONTEXT OF A WORD IN COMMON
ENGLISH CONTRIBUTES CONSIDER-
ABLY TO OUR ABILITY TO DECIPHER
IT FROM THREE LETTERS AND THE
LENGTH.

HOWEVER, IN COMPUTER PRO-
GRAMS THE CONTEXT IS LESS
CONFINING (COMMONLY, WORDS CAN
BE SWAPPED WITHOUT CHANGING THE
SEMANTICS). CONSEQUENTLY IT
CONTRIBUTES CONSIDERABLY LESS
HELP IN IDENTIFYING THE DEFINITION
OF A WORD. IN FACT, AS THIS LETTER
DEMONSTRATES, THREE LETTERS AND
A LENGTH CAN LETDOWN EVEN A
CAREFUL READER OF COMMON
ENGLISH.

SINCERELY YOURS,
ARTHUR GOLDBERG
SPENCER SOO-HOO

Your letter and its "translation" certainly make the point.—ed.

Dear Fig:

Right now I am trying to put together a local Danish FIG, and I would therefore like you to update me with the names and addresses of the Danish FIG members and possible make a note in FORTH DIMENSIONS about my intentions.

As the communication lines are rather long and since our magazine is only bimonthly, please inform me on your next deadline as soon as possible.

Robert L. Demrow
P. O. Box 158 Blustar
Andover, MA 01810

Thanks for the input. Your request has

Niels Oesten
Brostykkevej 139
DK-2650 Hvidovre
Denmark

Thanks Niels. Good luck on establishing a local Danish FIG Group. Anyone interested, please contact Niels as listed above. Regarding deadlines: Copy must be in our hands 6 weeks prior to publication, i.e., 4/15 is the deadline for May/June edition, etc.—ed.

Dear Fig:

I just wanted to write to tell you how much I enjoy FORTH DIMENSIONS. Every issue has several things of interest to me, and I appreciate your work in seeing that it gets done (often a thankless task). Here in New Hampshire, Rob Moore of SNAC (Southern New-Hampshire Apple Corps) is doing most of the work in implementing and refining a version of fig-FORTH for the Apple II. We have taken, as much as possible from page zero so that we can use the many subroutines available from the Applesoft ROM. I have been working with our version for some time now and am doing a high-resolution graphics game using FORTH and Applesoft hi-res routines.

Gregg Williams
BYTE Publications
PO Box 372
Hinscoca, NH 03449

Thanks Gregg. Glad you enjoy and appreciate our efforts.—ed.

Dear Fig:

Regarding the 8080 Renovation Project's requests for bug fixes, I would like to counter with a request that they provide a status report in FORTH DIMENSIONS that includes those bugs already reported along with any solutions proposed or implemented. It would also be of interest to find out what the goals are for the 8080 Renovation Project and how local FIG chapters can help.

There is what I consider a bug in that the message routine uses an absolute value of screen 4 and 5 for getting error message information. This is fine where offset is zero but when an offset other than zero is used and the disk has other information on absolute screens 4 and 5, things don't look too good.

Thanks for the input. Your request has

Robert L. Demrow
P. O. Box 158 Blustar
Andover, MA 01810

Thanks for the input. Your request has
Dear Fig:

After receiving my installation guide, I spent a week keying in the requisite seventy-five pages of 6502 assembler code. Now, I am a confirmed figger. I have written my own 6502 FORTH assembler, a small wordprocessor (with which I am writing this letter), and an APPLE II graphics utility resembling LOGO. I like FORTH!

Bob Wiseman
L18 St. Andrews Drive
Cincinnati, OH 45245

Your last name says it all: How about some articles for FD-publication?--ed.

Dear Fig:

I am spreading the FORTH gospel here in Taiwan. I have written about 20 lectures and numerous demonstrations to various universities and institutions in this area and have generated quite a bit of interest. At a meeting on October 25th attracted 20 enthusiasts. I am having fun too!

From ground zero, I can now count about 10 FORTH systems installed already. Many of them, from the FIG Installation Manual and source listing, I brought with me.

An informal FORTH discussion group has already been formed, and our last meeting on October 25th attracted 20 enthusiasts. I am having fun too!

My home phone in Taipei is 393-1554. If any of you happen to be heading this way, be sure to let me know.

Dr. Chen-hanson Ting
National Yang Ming Medical College
Taipei, Taiwan, 112 R.O.C.
(02) 831-2301

Glad to hear you are doing so well. We all miss you here.--ed.

Dear Fig:

I want to extend my apologies to you, your readers and the FORTH Interest Group. I sent in an announcement about FORTH ROMS for the TRS-80, MOD a several months ago. Unfortunately, circumstances beyond my control now force me to revise their availability schedule. I will spare everyone further embarrassment by waiting to send in another announcement until I have the first chip set in hand.

Martin Schaaf
P.O. Box 1001
Daly City, CA 94017

Thanks for the update, Martin. This points up why we have our policy of not announcing unreleased products.--ed.

Dear Fig:

In the packet of materials I got when I joined FIG was a copy of FORTH DIMENSIONS, Volume III, #1 with a product review of Timin-FORTH. The review interested me in two respects. One was the benchmark tests that it contained. The other were the comments on the alleged lack of superiority of the Z-80 compared to the 8080.

Those benchmarks gave me a chance to compare my machine with my version of FORTH—the results of which surprised me since my machine only runs at 2 mhz and the machine used in the review runs at 6 mhz. I expected my machine would take three times longer but in all tests; it ran comparable or even faster. It would seem my implementation is faster for involved arithmetic operations.

The editors of FORTH DIMENSIONS are right. I believe, in being wary of timing benchmarks for it is easy to draw invalid conclusions from them. In fact, the editors themselves drew the wrong conclusions!

The tests do not show the Z-80 runs benchmarks slower than the 8080—little Z-80 was used for both tests. The correct conclusion is that some FORTH implementations are more efficient than others and that some versions on the market are terribly slow.

I am sure you get a flood of letters following a benchmark; but I just had to write to say that the speed of FORTH is not necessarily just a function of processor speed as you have often claimed.

Everett Carter
Harvard University
Division of Applied Sciences
Cambridge, MA 02138

Thank you for your well thought out contribution.--ed.

Dear Fig:

A comment about your publication FORTH DIMENSIONS. The information is certainly useful (especially the applications), however, much of the material assumes a complete understanding of FORTH II inner workings. I don't really understand how the compiler works and what all those cryptic words (CA, PFA, SMUDGE, IMMEDIATE, etc.) do, but I would like to learn. I am sure there are others out there like me, so how about some tutorial articles on some of these FORTH-unique features.

Thomas Kastner
7918 207th St, SE
Snohomish, WA 98290

Entry-level tutorial articles are an area I have been exploring for the past year. Check this issue and you will find the first in a series of articles contributed by Henry Laxen of LAXEN AND HARRIS, INC., a firm that specializes in FORTH instruction.--ed.

Dear Fig:

I don't doubt FORTH would be more useful if my machine had all of the features described in "An Open Response" (Volume II, #6) but consider what I have gained without them:

1. An understanding of how FORTH works.
2. A demonstration that a workable subset of FORTH can be implemented on a very small system.
3. Hours of enjoyment and appreciation of FORTH's virtues.
4. A useable language faster and better than Tiny Basic and more convenient than machine language.
5. The ability to install and interactively test a larger version of FORTH when I expand my machine.

I feel the article does users of small systems a disservice. Instead of discouraging users of small systems, FIG should encourage development of standardized subsets for use on small machines.

Roger L. Cole
395 Elm Park Avenue
Elmhurst, IL 60126

FORTH DIMENSIONS publishes articles to encourage communications, thought and growth of the FORTH world. Far from discouraging users of small systems, the FIG leadership is composed almost exclusively of members with small systems. FIG has been a leader in encouraging the development and use of FORTH on small systems. In fact, it is probably safe to say that if there were no FIG, there would be no FORTH, for small users today. Most of the vendors and systems now in use have been derived from FIG listings provided at cost. The source data for these listings, FORTH DIMENSIONS, research, standards, etc., which so many take for granted, have been derived from the labor and cash contribu-
Dear Fig:

Congratulations to all the people who produce FORTH DIMENSIONS on its quality and improvement. Please send me a writer's kit so I can make some of my applications presentable for publication.

Bob Royce  
Box 57 Michiana  
New Buffalo, MI 49117

Your kit is on the way! Anyone else?

--ed.

Dear Fig:

Glen Haydon's nice article in FORTH DIMENSIONS III/2, page 57 talks about an algorithm he would like to have to determine the Julian day. With the background that FORTH has in astronomy, I'm sure there must be several, but this is the nicest I know. It comes from the U.S. Naval Observatory via an article in the Astrophysical Journal Supplement Series, Vol. 41 No. 3 Nov. 1979 pp 391-2.

Example: 3 20 1982 JD D.

Example: 3 20 1982 JD D.  
2453403 OK

If you are only concerned with dates between 3/1/1900 and 2/28/2000, then you can omit line 3 entirely.

On another subject, there is another correction I noticed in the dump of the fig-FORTH 6502 Assembly Source - at location OC32, 80 1A should be D7 0B.

Peter B. Dunckel  
52 Seventh Avenue  
San Francisco, CA 94118

Really slick! But the algorithm would be hard to explain to most people.--ed.

FUNCTIONAL PROGRAMMING AND FORTH

Harvey Glass  
University of South Florida  
College of Engineering  
Department of Computer Science  
Tampa, FL 33620

The distinguished computer scientist, John Backus, in his 1977 Turing Award lecture (1) describes the shortcomings of conventional programming languages and suggests a new approach to programming in a style described as functional programming (FP). We will summarize the faults that Backus finds in conventional languages, briefly describe the functional programming style, and lastly show that FORTH meets the spirit of this style of programming.

Conventional Languages

An underlying problem of conventional programming languages is that they tend to be high level descriptions of the Von Neumann computer. The assignment statement is the principal construct of these languages. A program becomes a series of these assignment statements, each of which requires the modification of a single cell. We may think of the Von Neumann computer as a set of storage cells, a separate processor, and a channel connecting the two. If assignment statements imitate the store operation, then branch statements imitate jump and test while variables imitate storage cells. The high level languages provide sophisticated constructs to directly model the underlying Von Neumann design. Conventional languages in the "word at a time" flow described above require large data transfers through this small channel connecting main storage and the CPU. Backus calls this the Von Neumann bottleneck. It is not merely a physical bottleneck but, more importantly, it is a bottleneck to our thinking about computer languages. The high level languages are not merely a physical bottleneck but, more importantly, it is a bottleneck to our thinking about computer languages.

Backus says that, "FP (Functional Programming) systems are so minimal that some readers may find it difficult to view them as programming languages. We have a set of predefined functions in a library (dictionary) and may define new functions in terms of these predefined functions. Functional forms are constructs denoting functions which take functions as parameters. For example, the construct "if-else-then" and the construct "do while" are functional forms. As indicated above, composition is also a functional form.

FORTH of course has predefined constructs which serve as the functional forms of FP systems. In fact, FORTH provides facilities for adding new functional forms. An example would be a "case" construct to provide a more flexible and clear decision structure than that of a set of nested "if-else-then"s. The capability of language to add new functional forms is not inherent in FP systems. Backus defines a language with this capability as a formal functional programming (FFP) language.

An Example of Functional Programming

The Factorial Function

An example of a program written in the style of functional programming is as follows:

def = eq 0 - 1 ; * [ ( ) , 2 subj ] , where the notation o * [ ] denotes functional forms. As we have seen, o denotes composition. The notation [f, f] denotes construction where [f, f] applied to an argument x yields the sequence f(f( (x), f(x)) ). The notation p f applied to an argument x indicates that the value p(x) is to be examined and if p(x) is true the expression yields f(x) else it yields g(x).

Other definitions used in the above are:

eq applied to x yields a value true if x is 0, and yields false otherwise.

I is the literal value 1 and yields the
value 1, regardless of the argument.
* is the multiplication operator, and applied to a sequence \( <x,y> \) yields \( xy \).
id is the identity operator. id applied to \( x \) yields \( x \).
subt applied to an argument \( x \) yields \( x-1 \).

Following the logic of the above function we see that \( \leq \) applied to an argument \( n \) yields 1 if \( n \) is zero. If \( n \) is not zero we generate

\[ \text{The same is true of some of the FORTH nouns like BLOCK, BUFFER, STATE, LIMIT, and BASE.} \]

Furthermore, use of a word in only one form rules out its use in another form, except where it can reside in a different vocabulary. Thus words like KEY, LOOP, BLANK, and TYPE (all FORTH verbs) can be used as verbs (questioning our temptation to use them that way for their inherent (English language-based) clarity. The same is true of some of the FORTH nouns like BLOCK, BUFFER, STATE, LIMIT, and BASE.

Thus it is not possible to know the nature of the word from its name alone. Would prefixes for verbs unnecessarily clutter the language? Would some prefix or suffix to differentiate constants from variables be useful? Or should we leave it alone. The TO and FROM words help clarify things but are not without problems, whereas @ and # are perfectly uniform in function. Could a FORTH-like language be built that allows the word-type to become part of the header, with the compiler choosing which form of the same-named word to use based on its syntactical position, like nouns (variables, constants, array) being objects of TO and FROM?

Summary
This very short summary of the article by John Backus does not begin to do justice to either the scope or depth of the paper.

The "new" type of programming has generated considerable interest within the computing community and most particu-
FORTH STANDARDS CORNER

Robert L. Smith

More Words on WORD

In my last column, I discussed WORD. I neglected to mention an important topic relating to the implementations of WORD which may influence transportability. Prior to the 79-Standard, the execution of WORD caused the string from the input medium to be moved to the dictionary area, starting at HERE with the character count. Some implementers would be tempted to define the 79-Standard WORD from the older WORD in a manner somewhat like this:

: WORD WORD HERE ;

Other implementers would probably put the string elsewhere. Now suppose that the user wishes to reverse the character string and replace the modified string in the dictionary. The result from the former implementer's system will not be as expected, and will not result in "equivalent execution" on the later implementer's system. A similar but more serious problem occurs with PAD. PAD is conventionally offset from HERE by a fixed amount (68 bytes in Fig-FORTH). There are at least three different solutions:

1. Implementations which place the string at HERE could be considered non-standard, and the problem goes away.

2. A clarification could be added to the Standard indicating either that the string will always be at HERE, or that it may be at HERE.

3. The problem could be forced upon users by requiring that the characters from WORD be stored in a user-defined area prior to their movement to the final destination.

Let Me Number the Ways

In many areas the 79-Standard defines limits and formats in painful detail. There is an important area in which very little is said, namely the format for single and double precision numbers in the input stream. In the section "Interpreter, text" it is clear that "numbers" are allowed in the input text stream and may either be compiled or placed on the parameter stack. A definition of the format of a number should include at a minimum the distinction between double and single precision, the sign of the number, and the set of allowed characters from which the number is constructed. In keeping with the spirit of the rest of the Standard, I would like to propose a few definitions which should be fairly easy to implement and which appear to be compatible with most current implementations (including Fig-FORTH). First, we define a digit:

**digit**

A digit is any one of a set of ASCII characters which represent numeric values in the range from 0 to base-1. For bases greater than decimal 10, the set of characters is 0 ... 9 A B C ... where the ascending ASCII sequence is used for A and above.

Next, we add to the original definition of number as follows:

**number**

A number is represented in the input stream as a word composed of a sequence of one or more digits with a leading ASCII minus (-) if the number is negative and a trailing ASCII dot (.) if the value is to be considered double precision.

I recommend that implementers allow the above format, and that authors of transportable programs adhere to the same format. In any case, when the Standards Team meets again, they should certainly clarify this area.

**Under the Spreading FIG-TREE**

As many of you are aware, there is a Computer Conference Tree (now nicknamed the FIG-TREE) which contains items of interest to the FORTH community. I would like to encourage all persons interested in the 79-Standard to read and contribute to the branch of the FIG-TREE called 79-STANDARD. All you need is a terminal (110 or 300 baud), a modem, and a telephone. The number is (615) 538-3580. See back issues of FORTH DIMENSIONS for further information, or just call up and send a few carriage returns until the system responds.

CORRECTIONS

Add to: FD III/4, pg. 102 the following:

REFERENCES


EDITOR'S NOTE:

Peter Bengtson of DATATRONIC AB in Stockholm, Sweden sent us a copy of the September, 1981 edition of Electronics And Computing Monthly. Feature article was FORTH, "The Language of the Eighties" in which FIG is mentioned prominently. More confirmation we are all riding the crest.

Change: FDIII/4, pg. 118, para 3 to:
The TO concept was developed by Dr. Paul Bartholdi as an alternative to constants and variables.
I have three questions about FORTH:

Q1. I know of two CP/M FORTHeRs that have their own way of dealing with the BIOS and BDOS and as a result cannot read each other's screens. What I'm leading to is this: CP/M and fig-FORTH are both supposed to be machine independent systems but cannot read each other's source code files. CP/M figures ought to get together on this one.

A. Differences between disk organizations are sector skewing and location. It is easy to add definitions to a FORTH which uses BIOS so it can read other organizations; it is not possible the other direction.

2. When selecting a new drive, you need to do a COLD start or you'll remain on the last drive--this is only true if you are accessing the same screen number. If you leave an empty line between two definitions on the screen, a LOAD will stop loading at the empty line. Are these FORTH conventions I haven't heard about yet or are they peculiar to my Timin FORTH?

A. Both of these are bugs--demand fixes from Timin.

3. Somehow, I've been leaving a lot of control characters behind when using the editor. They don't show up on a screen list but they sure ruin any attempt at loading the screen. I am not sure if this is a common problem but I have enclosed a short routine to replace control characters with spaces for anyone else who has this problem.

SCREEN: 95

('HUNT FOR CONTROL CHARACTERS')
'HUNT (SCREEN #---)

BLOCK
1024 0 DO DUP C@ DUP 32 < IF CR, CR, 66 + EMIT
" @ ? DUP U, ELSE DROP
ENDF 1> LOOP DROP;

'FIXSCREEN (SCREEN #---)

BLOCK
1024 0 DO DUP C@ DUP 32 < IF DUP 32 SWAP @ ENDIF
1+ LOOP DROP;

'ACTUALLY HUNT AND FIXSCREEN ARE QUITE SIMILAR, HUNT JUST SHOWS UP ANY GLIBLY CHARACTERS AND FIXSCREEN REPLACES THEM'

A. Don't know. May be an editor bug or the way you are using it. If you add a line with #P followed immediately by a carriage return in the fig editor, a null is introduced into the line which stops compiling. (editor fix should be supplied)

THAT MYSTERIOUS fig-FORTH AMNESIA

Many fig-FORTH users have probably noticed the curious phenomenon I refer to as "amnesia" in their computers, and those who understand the method of the fig-FORTH dictionary search, no doubt understand it as well. It is an amusing, often perplexing, but usually useful property peculiar to fig-FORTH dictionaries.

Because names in fig-FORTH may have variable length, the distance between the start of the name and the link to the next name in the dictionary is also variable. Because the width (number of characters saved) is also allowed to be less than the actual number of characters in the name, one cannot rely on the count to provide the address of the link-field, given the address of the name-field. This is why the fig-FORTH compiler automatically sets the most significant bit of the first character and the last character in every name. By this device, one can scan a name forward or backward by looking for this bit.

In a dictionary search, the address in the link-field is followed to the beginning of the name-field of the previous word. If it is not a match to the key you are looking for, we scan forward in memory until the most significant bit tells us we have found the link-field to the next word. When a dictionary link is "broken" by cropping RAM, an erroneous address is followed, and the system is said to "crash".

However, in fig-FORTH, the system does not always "die". In many cases, it is merely "wounded", displaying a strange kind of amnesia in which it has no recollection of recent definitions, but remembers with clarity its "childhood". What happens is this: the broken link sends the dictionary search off to a totally random part of memory (if you do not have 64K, it may address RAM where there are no boards). Since it is not likely to find a match at this address, it scans forward for the most significant bit that marks the end of the "name". The odds are that it will eventually find one, mistake the next two bytes for a link, and follow another wild address somewhere else.

Now, depending on how much of your memory is filled with dictionary, and depending on what is in your unused RAM, the odds are not bad that after bouncing aimlessly around for awhile, the search may land in the middle of a valid name. One does not expect a match to compare with the middle of a name, but the search then scans for the most significant bit, finds a valid link, and gets back into the dictionary. What the "amnesia" has actually forgotten, then, is everything between the broken link and the point where the search re-enters the dictionary.

If your used RAM is large in comparison to FORTH, you are likely to find most of FORTH still available as a kind of crippled monitor to help you find out what went wrong without re-booting the system (which destroys the damage). Furthermore, since you now know the cause of this illness, you can exploit it to your advantage. Simply modify your boot-up RAM-check routine so that it leaves a pattern in your unused RAM, such that no matter how it is viewed, it will appear to be an address somewhere in the middle of a name-field, somewhere near the top of your basic FORTH and utilities. You will now find, to your delight, that when you "crash", you usually have your most powerful tools still at your disposal.

Users of FORTH, Inc. Micro-FORTH are not likely to observe this phenomenon. Because names are always exactly four characters long, the link field does not have to be scanned for; instead, it is found by simple arithmetic. In order to re-enter the dictionary, one must land by chance on the exact beginning of a name-field. Much more likely than this, is that the search will enter a loop in which it goes again to an address it has already visited, and get caught forever. Remember that the addresses found are by no means random. All you have to do is cover the most common ones.

Steve Munson
8071 E. 7th Street, #14
Buena Park, CA 90621

TRANSIENT DEFINITIONS

These utilities allow you to have temporary definition (such as compiler words: CASE, OF ENDOF, ENDCASE, GODO, etc.) in the dictionary during compilation and then remove them after compilation. The word TRANSIENT moves the dictionary pointer to the "transient area" which must be above the end of the current dictionary. The temporary definitions are then compiled into this area. Next, the word PERMANENT restores the dictionary to its normal location. Now the application program is compiled and the temporary definitions are removed with the word DISPOSE. DISPOSE will take a few seconds because it goes through every link (including vocabulary links) and patches them to bypass all words above the dictionary pointer.

NOTE: These words are written in MicroMotion's FORTH-79 but some non-79 Standard words are used. The non-Standard words have the fig-FORTH definitions.

Philip Wasson
In analyzing or proposing changes to any Standard definition, it is very important to concentrate on the details of the needed function and to avoid any preconceived notion of internal implementation details, unless, of course, the two are inseparable. If this is not done, we can severely and unnecessarily constrain future implementors from doing their best possible job, or, worse yet, find them avoiding the Standard entirely.

A good case in point is the word WORD. Since most FORTH implementors have favored using the "free space" above the dictionary to store tokens extracted by WORD, and further since their experience seems to be centered around small to medium sized application programs, it is not surprising that this free space is arbitrarily large. In addition to storing tokens at HERE, PAD is usually also defined to float above the dictionary in this "unbounded" free space. Therefore, whether WORD handles tokens of length 128, 256 or even 1024 bytes is innocently discussed with the idea that the only issue involved is the length descriptor preceding the string!

However, whether this token buffer and PAD float above HERE or are fixed location buffers or some different scheme is devised, they consume real memory and are not really "free space". To illustrate, suppose we assume the traditional implementation for a moment and use HERE as the start of the token buffer used by WORD. The PAD is then usually floated at a location equal to HERE plus some constant. If WORD must handle tokens as long as 255 bytes, then PAD must be floated at least 256 bytes above HERE to prevent token extraction from corrupting the contents of PAD. The 79-STANDARD requires that PAD be able to hold at least 64 bytes, so now we're at HERE + 320 bytes.

If one is compiling a large application program, the dictionary will grow until eventually HERE + 320 hits the peg (whether it is a fixed boundary or the PSTACK bottom or whatever). When it does, no more compilation can take place (even though there is at least 320 bytes of unused dictionary left) without violating the Standard. If you permit further compilation, the size of PAD begins to drop below the minimum 64, which is not allowed. Even if you start automatically reducing the PAD offset so that it remains fixed in size, the token buffer begins shrinking and can no longer satisfy the 256 byte string requirement.

I'm trying to illustrate that "free space" is only "free" as long as all of memory isn't needed. When memory fills, these "free space" buffers prevent code from being compiled into them. The floating buffer concept seems to obscure this fact more than if the token buffer and the PAD were given fixed, dedicated areas of memory.

If the token buffer must handle 1024 byte strings, the situation is even worse. We then have to stop compiling when the dictionary has over 1K bytes of space left! Since most of the time the tokens extracted by WORD are very short (31 characters or less), we pay a dear price to be able to handle the occasional long string, given that WORD must handle it, and WORD is defined as at present.

If you discard the notion that a more or less unbounded "free space" exists somewhere in memory, the approach to WORD's definition takes on a new facet. At Microsystems, we have developed several large applications using FORTH, which resulted in target compiled code in the range of 32K to 48K bytes, exclusive of the dictionary headers and the FORTH operating system software. When applications become that large, there isn't even room to hold all the names in memory at one time (even if constrained to 3 characters and length), let alone room to burn for large "free space" buffers! Our implementations, which is called proFORTH™ handles this problem by means of multiple dictionaries and ROM/RAM segment control with selective symbol purging. Names are classified as to their needed lifetimes during compilation. When the names are no longer needed, they are purged and their memory space is reclaimed. This allows much of the memory devoted to dictionary headers to be reused many times during compilation, thereby enabling very large applications to be compiled.

The foregoing is not a commercial for proFORTH, but rather is intended to illustrate that the scope of usage to which FORTH can be applied is very broad. In a situation where you have multiple dictionaries and are fighting for every byte of memory available, thinking in terms of storing unbounded tokens at HERE and floating PADs of arbitrary length becomes very incongruous. Admittedly, I've described a somewhat extreme situation, but it is not as rare as you may think. Microprocessor applications are getting more ambitious every day and sooner or later you will have a crowded memory condition. I think FORTH should be able to handle these situations gracefully, without having to deviate from the Standard.

When defining WORD, then, one objective should be to enable users to extract arbitrarily long tokens from the text stream but not force the implementor to provide an arbitrarily long memory buffer to accomplish it. While this may sound a little like trying to "have your cake and eat it too", a rather simple factoring of WORD can easily accomplish it. To illustrate my point, suppose we devise a more basic WORD called (WORD) and define it as follows:

```
: WORD ( c -- a )  HERE @ DUP IF \BLK
  BLK \STK @
THEN 2PK @ + SWAP ENCLOSE
>IN *\ OVER +P +SWAP ;
```

where ENCLOSE is defined as in the FIG glossary and -ROT is equivalent to ROT ROT.

This new (WORD) extracts the next token from the text stream, delimited by c, and leaves its address and length on the stack. Actually, the token is merely left in the input buffer (keyboard or disk) and a pointer to it is given. Thus, no additional or temporary buffer is needed. The user may now do anything he (she) wants with the string, including moving it to HERE if desired (and if it will fit).

For example, if you want to compile the token as a "dot-quotate" string, a definition such as WORD, can be used:

```
: WORD ( c -- )  HERE OVER + P: bi DUP SWAP OVER C: COUNI
COUNT COUNI ;
```

If you want a blank-filled line put in PAD, the following could be used:

```
: TEXT ( c -- )  PAD/\ C: \ELEMS BLANKS
(WORD) C: Min PAD C: 1 PAD COUNT COUNI ;
```

For the routine compiler/interpreter job of extracting small (31 characters or less) tokens from the text stream, the following could be used:

```
: WORD ( c -- )  HERE OVER + P: bi DUP SWAP OVER C: WBF
WBF C: WBF C: WBF COUNT COUNI DUPE SWAP COUNI ;
```

where WBFR is a "small" word buffer limited to WDSZ + 2. Note that except possibly for the self-imposed size limitation, the last definition satisfies the 79-STANDARD definition of WORD.

If you will carefully examine these constructs, you can quickly discover that given (WORD) as the elementary form, the user can extract tokens of any size, put them wherever he wants, and format them with or without the trailing delimiters, or for that matter, the leading count byte (or 16 bit word if you prefer). In other words, the user ought to be able to do essentially anything that he may desire, but, the implementor need not provide any special, temporary buffers or arbitrary size just to
satisfy the Standard.

Using (WORD) as the fundamental token extractor allows implementors to compile dot quote strings, for example, without the need for any transitional buffers (see (WORD)). On the other hand, if dot quote strings are acquired by the present form of WORD in the Standard, then the token buffer must be at least as large as the longest dot quote string, which is presently specified to be 127 characters.

One might argue that if the buffer is at HERE, there is no penalty since that is where the string must go anyway, and if it won't fit it can't be compiled. However, this line of reasoning is again limited by a parochial view that all FORTH implementations must be alike. If a system like proFORTH is being used, the target definition body can optionally be compiled "in place" separate from the dictionary header. There may be room for the string in the target segment of memory but not enough in the dictionary.

In conclusion, let me say that if there is sufficient memory, the user may declare all the buffers he wants, but we should not require that these buffers be preallocated by the implementor in order to satisfy the Standard. Therefore, I submit that my definition of (WORD) is a more fundamentally valuable function than WORD (as currently defined in the 79-STANDARD,) from which all others can be built without burning sometimes precious memory space. There are already enough buffers and such required (directly or indirectly) by the Standard. Let's not arbitrarily insist on more by accidently defining words in such a way as to force an implementor to provide them.

* I emphasize "possibly" because fortunately the Standard is not explicit as to the length of tokens that must be handled by WORD.

CORRECTION TO FEDIT

Sorry you had trouble with FEDIT. The listing was retyped at FIG and several typos crept in. They are:

1. SCR 64 Line 10: compile should be COMPIL.
2. SCR 65 Line 23: 1+/MOD should be 1+16/MOD
3. SCR 67 Line 48: B/BUD should be B/BUF
4. SCR 67 Line 49: ie should be :IE
5. SCR 67 Line 50: ALIN should be +ALIN

You are perfectly right that source text should be loadable. I talked to some of the people at FIG about this and they were acutely aware of the problem but they are simply not set up to directly reproduce listings into F0 at the present time. They do the best job they can with the resources available to them, and they work darn hard at it. I can't fault them.

REPL is a pseudonym for the FIG-FORTH line editor definition, R, I used the pseudonym because FEDIT was the first program I wrote in FORTH and I really wasn't familiar enough with Vocabulary's to comfortably use a word that was already used in the FORTH vocabulary.

Let me know how it works for you. If you would like a machine produced listing, I can run one for you from my current version.

Edgar H. Fey, Jr.
18 Calendar Court
La Grange, IL 60525

A HELPFUL UTILITY

Here's a short FORTH word of great utility that I use heavily in my screens. I hope you like it. Its name is CVD, which stands for "convert to decimal".

DECIMAL
: CVD
BASE 10 SWAP
OVER 10/ MOD
10 * +
10 * +
;

I like to work in hexadecimal, but often make mistakes when using the words LOAD, LIST, and many of the FORTH screen editor words because I'm thinking in decimal when the system's in hex. And I don't do the following:

: LIST CVD LIST

then 130 LIST lists screen 130 whether I'm in decimal or hex. It also works for any other base, as long as that base accepts the number.

As to how it works, a little work will show that CVD splits a three-digit number into its respective digits (IE, 130 becomes 1, 3, and 0) and reassembles the digits into the number that is, in decimal, the same as the keys pressed by the user.

Gregg Williams
BYTE Publications
PO Box 372
Hanock, NH 03449

CALL FOR PAPERS

1982 Rochester FORTH Conference on
Data Bases and Process Control
May 17 through May 21, 1982
University of Rochester
Rochester, New York

The second annual Rochester FORTH Conference will be held in May, and will be hosted by the University of Rochester's Laboratory for Laser Energetics. This year's topics complement and extend the work described at the 1981 FORML Conference and the previous Rochester Conference. We believe that the areas of data bases and process control can be uniquely dealt with using FORTH.

There is a call for papers on the following topics:

1. Data Bases, including, but not limited to: hierarchical, network and relational models; scientific use; process control; and commercial systems.
2. Process Control, including, but not limited to: multitasking, meta-compilation, data acquisition and real time systems; video games.
3. Related concepts of: implementation, speed/space tradeoffs; user interactions; designer tools; and graphics.

Papers will be handled in either oral sessions or poster sessions, although oral papers will be refereed in accordance with conference direction, paper quality and topic. Please submit a 200 word abstract by March 15, 1982. The oral papers deadline is April 15, 1982, and the poster papers deadline is May 1, 1982. Send abstracts and papers to the conference chairman, Lawrence Forsley, by those dates. Please keep papers to a maximum of 10 printed pages. If this restriction causes a serious problem, contact us.

For more information, please contact the conference chairman at:

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

La Grange, IL 60525

FIGURE DIMENSIONS III/5
A FORTH ASSEMBLER FOR THE 6502
by William F. Ragdsdale

INTRODUCTION

This article should further polarize the attitudes of those outside the growing community of FORTH users. Some will be fascinated by a label-less, macro-assembler whose source code is only 96 lines long! Others will be repelled by reverse Polish syntax and the absence of labels.

The author immodestly claims that this is the best FORTH assembler ever distributed. It is the only such assembler that detects all errors in op-code generation and conditional structuring. It is released to the public domain as a defense mechanism. Three good 6502 assemblers were submitted to the FORTH Interest Group but each had some lack. Rather than merge and edit for publication, I chose to publish mine with all the submitted features plus several more.

Imagine having an assembler in 1300 bytes of object code with:
1. User macros (like IF, UNTIL,) definable at any time.
2. Literal values expressed in any numeric base, alterable at any time.
3. Expressions using any resident computation capability.
4. Nested control structures without labels, with error control.
5. Assembler source itself in a portable high level language.

OVERVIEW

Forth is provided with a machine language assembler to create execution procedures that would be time inefficient, if written as colon-definitions. It is intended that "code" be written similarly to high level, tested, and executed by Forth. CODE builds the following name "MON" into a dictionary header and calls ASSEMBLER as the CONTEXT vocabulary.

As a practical example, here’s a simple call to the system monitor, via the BRK opcode.

CODE MON ( exit to monitor ) BRK, NEXT JMP, END-CODE

The word CODE is first encountered, and executed by Forth. CODE builds the following name "MON" into a dictionary header and calls ASSEMBLER as the CONTEXT vocabulary.

The "" is next found in FORTH and executed to skip till ".". This method skips over comments. Note that the name after CODE and the "" after "" must be on the same text line.

CODES

BRK, is next found in the assembler as the op-code. When BRK, executes, it assembles the byte value 00 into the dictionary as the op-code for "break to monitor via NMI".

Many assembler words names and in ".". The significance of this is

1. The comma shows the conclusion of a logical grouping that would be one line of classical assembly source code.
2. "." compiles into the dictionary; thus a comma implies the point at which code is generated.
3. The "" distinguishes op-codes from possible hex numbers ADC and ADD.

NEXT

Forth executes your word definitions under control of the address interpreter, named NEXT. This short code routine moves execution from one definition, to the next. At the end of your code definition, you must return control to NEXT or else to code which returns to NEXT.

RETURN OF CONTROL

Most 6502 systems can resume execution after a break, since the monitor saves the CPU register contents. Therefore, we must return control to Forth after a return from the monitor. NEXT is a constant that specifies the machine address of Forth’s address interpreter (say $0242). Here it is the operand for JMP,,. As JMP, executes, it assembles a machine code jump to the address of NEXT from the assembly time stack value.

SECURITY

Numerous tests are made within the assembler for user errors:

1. All parameters used in CODE definitions must be removed.
2. Conditionals must be properly nested and paired.
3. Address modes and operands must be allowable for the op-codes.

These tests are accomplished by checking the stack position (in CSP) at the creation of the definition name and comparing it with the position at END-CODE. Legality of address modes and operands is insured by means of a mask associated with each operand.

Remember that if an error occurs during assembly, END-CODE never executes. The result is that the "smudged" condition of the definition name remains in the "smudged" condition and will not be found during dictionary searches.

The user should be aware that one error not trapped is referencing a definition in the wrong vocabulary:

i.e., 0= of ASSEMBLER when you want 0= of FORTH
OP-CODES, revisited

The bulk of the assembler consists of dictionary entries for each op-code. The 6502 one mode op-codes are:

```
BRK, CLC, CLD, CLI, CLV,
DEX, DEY, INX, INY, NOP,
PHA, PHP, PLA, PLP, RTI,
RTS, SEC, SED, SEI, TAX,
TAY, TSX, TXS, TXA, TYA,
```

When any of these are executed, the corresponding op-code byte is assembled into the dictionary.

The multi-mode op-codes are:

```
ADC, AND, CMP, EOR, LDA, ORA, SBC, STA, ASL, DEC,
INC, LSR, ROL, ROR, STX, CPX, CPY, LDX, LDY, STY,
JSR, JMP, BIT,
```

These usually take an operand, which must already be on the stack. An address mode may also be specified. If none is given, the op-code uses z-page or absolute addressing. The address modes are determined by:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Mode</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>.A</td>
<td>accumulator</td>
<td>none</td>
</tr>
<tr>
<td>#</td>
<td>immediate</td>
<td>8 bits only</td>
</tr>
<tr>
<td>,X</td>
<td>indexed X</td>
<td>z-page or</td>
</tr>
<tr>
<td>,Y</td>
<td>indexed Y</td>
<td>z-page or</td>
</tr>
<tr>
<td>X</td>
<td>indexed indirect X</td>
<td>z-page only</td>
</tr>
<tr>
<td>Y</td>
<td>indirect indexed Y</td>
<td>z-page only</td>
</tr>
<tr>
<td>none</td>
<td>memory</td>
<td>z-page or</td>
</tr>
</tbody>
</table>

Examples

Here are examples of Forth vs. conventional assembly. Note that the operand comes first, followed by any mode modifier, and then the op-code mnemonic. This makes best use of the stack at assembly time. Also, each assembler word is set off by blanks, as is required for all Forth source text.

```
.A ROL, ROL A
1 # LDY, LDY #1
DATA, X STA, STA DATA, X
DATA, Y CMP, CMP DATA, Y
6 X ADC, ADC (06, X)
POINT, Y STA, STA (POINT), Y
VECTOR ) JMP, JMP (VECTOR)
```

(A distinguishes from hex number 0A)

The words DATA and VECTOR specify machine addresses. In the case of "6 *X ADC," the operand memory address $0006 was given directly. This is occasionally done if the usage of a value doesn't justify devoting the dictionary space to a symbolic value.

6502 Conventions

Stack Addressing

The data stack is located in z-page, usually addressed by "Z-PAGE,X". The stack starts near $0000E and grows downward. The X index register is the data stack pointer. Thus, incrementing X by two removes a data stack value; decrementing X twice makes room for one new data stack value.

Sixteen bit values are placed on the stack according to the 6502 convention; the low byte is at low memory, with the high byte following. This allows "indexed, indirect X," directly off a stack value.

The bottom and second stack values are referenced often enough that the support words BOT and SEC are included. Using

```
BOT LDA, assembles LDA (0,X) and 
SEC ADC, assembles ADC (2,X)
```

BOT leaves 0 on the stack and sets and the address mode to X. SEC leaves 2 on the stack also setting the address mode to X.

Here is a pictorial representation of the stack in z-page.

```
<table>
<thead>
<tr>
<th>see high</th>
<th>sec low</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot high</td>
<td>bot low</td>
</tr>
</tbody>
</table>
```

<--X offset above $0000

Here is an examples of code to "or" to the accumulator four bytes on the stack:

```
BOT LDA, LDA (0,X)
BOT 1 ORA, ORA (0,X)
SEC ORA, ORA (3,X)
SEC 1 ORA, ORA (3,X)
```

To obtain the 14-th byte on the stack:

```
BOT 13 + LDA,
```

Return stack

The Forth Return Stack is located in the 6502 machine stack in Page 1. It starts at $01FE and builds downward. No lower bound is set or checked as Page 1 has sufficient capacity for all (non-recursive) applications.

By 6502 convention the CPU's register points to the next free byte below the bottom of the Return Stack. The byte order follows the convention of low significance byte at the lower address.

Return stack values may be obtained by: PLA, PLP, which will pull the low byte, then the high byte from the return stack. To operate on arbitrary bytes, the method is:

1)  save X in XSAVE
2)  execute TSX, to bring the S register to X.
3)  use RPP to address the lowest byte of the return stack. Offset the value to address higher bytes. (Address mode is automatically set to X.)
4)  Restore X from XSAVE.

As an example, this definition non-destructively tests that the second item on the return stack (also the machine stack) is zero.

```
CODE IS-IT ( zero ? )
XSAVE STX, TSX, (setup for return stack)
RPP 2 + LDA, RPP 3 + ORA, ( or 2 item's two bytes together)
0= IF, INY, THEN, ( if zero, bump Y to one)
TAY, PHA, XSAVE LDX, (save low byte, restore data stack)
PUSH JMP, END-CODE ( push boolean)
```

```
<table>
<thead>
<tr>
<th>Return Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>hi byte</td>
</tr>
<tr>
<td>lo byte</td>
</tr>
</tbody>
</table>

S-->
free byte
```

Here is an examples of code to "or" to the accumulator four bytes on the stack:
FORTH REGISTERS

Several Forth registers are available only at the assembly level and have been given names that return their memory addresses. These are:

1. IP address of the Interpretive Pointer, specifying the next Forth address which will be interpreted by NEXT.
2. W address of the pointer to the code field of the dictionary definition just interpreted by NEXT. W-1 contains $6C$, the op-code for indirect jump. Therefore, jumping to W-1 will indirectly jump via W to the machine code for the definition.
3. User Pointer containing address of the base of the user area.
4. A utility area in z-page from N-1 thru N+7.

CPU Registers

When Forth execution leaves NEXT to execute a CODE definition, the following conventions apply:

1. The Y index register is zero. It may be freely used.
2. The X index register defines the low byte of the bottom data stack item relative to machine address $0000$.
3. The CPU stack pointer S points one byte below the low byte of the bottom return stack item. Executing PLA, will pull this byte to the accumulator.
4. The accumulator may be freely used.
5. The processor is in the binary mode and must be returned in that mode.

XSAVE

XSAVE is a byte buffer in z-page, for temporary storage of the X register. Typical usage, with a call which will change X, is:

CODE DEMO
XSAVE STX, USER'S JSR, (which will change X)
XSAVE LDX, NEXT JMP, END-CODE

N Area

When absolute memory registers are required, use the 'N Area' in the base page. These registers may be used as pointers for indexed/indirect addressing or for temporary values. As an example of use, see CMOVE in the system source code.

The assembler word N returns the base address (usually $0001$). The N Area spans 9 bytes, from N-1 thru N+7. Conventionally, N-1 holds one byte and N, N+2, N+4, N+6 are pairs which may hold 16-bit values. See SETUP for help on moving values to the N Area.

It is very important to note that many Forth procedures use N. Thus, N may only be used within a single code definition. Never expect that a value will remain there, outside a single definition!

CODE DEMO HEX
6 $ LDA, N 1 - STA, (setup a counter)
BEGIN, 0001 BIT, (tick a port)
N 1 - DEC, (decrement the counter)
0= UNTIL, NEXT JMP, END-CODE (loop till negative)

SETUP

Often we wish to move stack values to the N area. The sub-routine SETUP has been provided for this purpose. Upon entering SETUP the accumulator specifies the quantity of 16-bit stack values to be moved to the N area. That is, A may be 1, 2, 3, or 4 only:

3 # LDA, SETUP JSR,

stack before N after stack after
H high H
G low F bot- G
F
E
D
C
B
A

CONTROL FLOW

Forth discards the usual convention of assembler labels. Instead, two replacements are used. First, each Forth definition name is permanently included in the dictionary. This allows procedures to be located and executed by name at any time as well as be compiled within other definitions.

Secondly, within a code definition, execution flow is controlled by label-less branching according to "structured programming". This method is identical to the form used in colon definitions. Branch calculations are done at assembly time by temporary stack values placed by the control words:

BEGIN, UNTIL, IF, ELSE, THEN

Here again, the assembler words end with a comma, to indicate that code is being produced and to clearly differentiate from the high-level form.

One major difference occurs: High-level flow is controlled by run-time boolean values on the data stack. Assembly flow is instead controlled by processor status bits. The programmer must indicate which status bit to test, just before a conditional branching word (IF, and UNTIL).

Examples are:

PORT LDA, 0= IF, 0 THEN,
(read port, if equal to zero do $<a>$)

PORT LDA, 0= NOT IF, $<a>$ THEN,
(read port, if not equal to zero do $<a>$)

The conditional specifiers for 6502 are:

CS test carry set
0K byte less than zero
0= equal to zero
CS NOT test carry clear
0= equal to zero
0= NOT test not equal zero

The overflow status bit is so rarely used, that it is not included. If it is desired, compiler

ASSEMBLER DEFINITIONS HEX
50 CONSTANT VS (test overflow set)

CONDITIONAL LOOPING

A conditional loop is formed at assembler level by placing the portion to be repeated between BEGIN, and UNTIL:

6 # LDA, N STA, (define loop counter in N)
BEGIN, PORT DEC,
( repeated action )
N DEC,
0= UNTIL,
(N reaches zero)

First, the byte at address N is loaded with the value 6. The beginning of the loop is marked (as assembly time) by BEGIN. Memory at PORT is decremented, then the loop counter in N is decremented. Of course, the CPU updates its status register as N is decremented. Finally, a test for Z=1 is made; if N hasn't reached zero, execution returns to BEGIN. When N reaches zero (after executing PORT DEC, 6 times) execution continues ahead after UNTIL. Note that
BEGIN, generates no machine code, but is only an assembly time locator.

**CONDITIONAL EXECUTION**

Paths of execution may be chosen at assembly in a similar fashion and done in colon-definitions. In this case, the branch is chosen based on a processor status condition code.

PORT LDA, O= IF, (for zero set) THEN, (continuing code)

In this example, the accumulator is loaded from PORT. The zero status is tested if set (Z=1). If so, the code (for zero set) is executed. Whether the zero status is set or not, execution will resume at THEN.

The conditional branching also allows a specific action for the false case. Here we see the addition of the ELSE, part.

PORT LDA, O= IF, <for zero set> ELSE, <for zero clear> THEN, <continuing code>

The test of PORT will select one of two execution paths, before resuming execution after THEN. The next example increments N based on bit D7 of a port.

PORT LDA, O= IF, N DEC, (if D=1, decrement N) ELSE, N INC, (if D=0, increment N) THEN, (continue ahead)

**CONDITIONAL NESTING**

Conditionals may be nested, according to the conventions of structured programming. That is, each conditional sequence begun (IF, BEGIN,) must be terminated (THEN, UNTIL,) before the next earlier conditional is terminated. An ELSE, must pair with the immediately preceding IF.

BEGIN, <code always executed>

CS IF, <code if carry set>
ELSE, <code if carry clear>
THEN, 0= NOT UNTIL, (loop till condition flag is non-zero)
<code that continues onward>

Next is an error that the assembler security will reveal.

BEGIN, PORT LDA, O= IF, BOT INC, O= UNTIL, ENDF,

The UNTIL, will not complete the pending BEGIN, since the immediately preceding IF, is not completed. An error trap will occur at UNTIL, saying "conditions not paired".

RETURN OF CONTROL, revisited

When concluding a code definition, several common stack manipulations often are needed. These functions are already in the nucleus, so we may share their use just by knowing their return points. Each of these returns control to NEXT.

POP POPTWO remove one 16-bit stack values.
POPTWO remove two 16-bit stack values.
PUSH add two bytes to the data stack.
PUT write two bytes to the data stack, over the present bottom of the stack.

Our next example complements a byte in memory. The bytes' address is on the stack when INVERT is executed.

CODE INVERT (a memory byte ) HEX BOT X) LDA, (fetch byte addressed by stack)
FF # EOR, (complement accumulator)
BOT X) STA, (replace in memory )
POP JMP, END-CODE ( discard pointer from stack, return to NEXT )

A new stack value may result from a code definition. We could program placing it on the stack by:

CODE ONE ( put 1 on the stack )
DEX, DEX, ( make room on the data stack )
1 # LDA, BOT STA, (store low byte)
BOT 1+ STY, (hi byte stored from Y since = zero)
NEXT JMP, END-CODE

A simpler version could use PUSH:

CODE ONE
1 # LDA, PHA, ( push low byte to machine stack )
TYA, PUSH JMP, ( high byte to accumulator, push to data stack )
END-CODE

The convention for PUSH and PUT is:
1. push the low byte onto the machine stack.
2. leave the high byte in the accumulator.
3. jump to PUSH or PUT.

PUSH will place the two bytes as the new bottom of the data stack. PUT will overwrite the present bottom of the stack with the two bytes. Failure to push exactly one byte on the machine stack will disrupt execution upon usage.

**FOOLING SECURITY**

Occasionally we wish to generate unstructured code. To accomplish this, we can control the assembly time security checks, to our purpose. First, we must note the parameters utilized by the control structures at assembly time. The notation below is taken from the assembler glossary. The --- indicates assembly time execution, and separate input stack values from the output stack values of the words execution.

BEGIN, ==> --- addrB 1
UNTIL, ==> addrE 1 cc ---
IF, ==> cc --- addrE 2
ELSE, ==> addrE 2 ---
THEN, ==> addrE 2 --- or addrF 2 ---

The address values indicate the machine location of the corresponding \`BEGIN, IF, or ELSE,\` cc represents the condition code to select the processor status bit referenced. The digit 1 or 2 is tested for conditional pairing.

The general method of security control is to drop off the check digit and manipulate the addresses at assembly time. The security against errors is less, but the programmer is usually paying intense attention to detail during this effort.

To generate the equivalent of the high level:

BEGIN <a> WHILE <b> REPEAT
we write in assembly:

BEGIN, DROP (the check digit 1, leaving addrB)
<y>
CS IF, (leaves addrI and digit 2)
<y>
ROT (bring addrB to bottom) JMP, (to addrE of BEGIN, ) ENDF, (complete false forward branch from IF,)

It is essential to write the assembly time stack on paper, and run through the assembly stops, to be sure that the check digits are dropped and re-inserted at the correct points and addresses are correctly available.

**ASSEMBLER GLOSSARY**

# Specify 'immediate' addressing mode for the next op-code generated.

SY Specify 'indirect indexed Y' addressing mode for the next op-code generated.
BEGIN, --- addr 1 (assembling)

ASSEMBLER in FORTH

0 < --- cc (assembling)

Specify that the immediately following conditional will branch based on the processor status bit being negative (Z=1), i.e., less than zero. The flag cc is left at assembly time; there is no run-time effect on the stack.

0= --- cc (assembling)

Specify that the immediately following conditional will branch based on the processor status bit being equal to zero (Z=1). The flag cc is left at assembly time; there is no run-time effect on the stack.

0< --- cc (assembling)

Specify that the immediately following conditional will branch based on the processor status bit being negative (Z=1), i.e., less than zero. The flag cc is left at assembly time; there is no run-time effect on the stack.

0= --- cc (assembling)

Specify that the immediately following conditional will branch based on the processor status bit being equal to zero (Z=1). The flag cc is left at assembly time; there is no run-time effect on the stack.

BEGIN, --- addr 1 (assembling) --- (run-time)

Occurs in a code definition in the form:

BEGIN, ... cc UNTIL,

At run-time, BEGIN, marks the start of an assembly sequence repeatedly executed. It serves as the return point for the corresponding UNTIL,. When reaching UNTIL, a branch to BEGIN, will occur if the processor status bit given by cc is false; otherwise execution continues ahead.

At assembly time, BEGIN, leaves the dictionary pointer address addr and the value 1 for later testing of conditional pairing by UNTIL.

END-CODE

An error check word marking the end of a CODE definition. Successful execution to and including END-CODE will unsmudge the most recent CURRENT vocabulary definition, making it available for execution. END-CODE also exits the ASSEMBLER making CURRENT the same as CURRENT. This word previously was named C;.

IF, cc --- addr 2 (assembly time) --- addr 2 (assembly time)

Occurs within a code definition in the form:

cc IF, <true part> ELSE, false part THEN,

At run-time, IF, branches based on the condition code cc, (0 or 0s) or 0s. If the specified processor status bit is true, execution continues ahead, otherwise branching occurs to just after ELSE, (or THEN, when ELSE, is not present). At ELSE, execution resumes at the corresponding THEN.

When assembling, IF, creates an unresolved forward branch based on the condition code cc, and leaves addr 2 and 2 for resolution of the branch by the corresponding ELSE, or THEN,. Conditionals may be nested.

INDEX --- addr (assembling)

An array used within the assembler, which holds bit patterns of allowable addressing modes.

IP --- addr (assembling)

Used in a code definition in the form:

IP STA, or IP Y LDA,

A constant which leaves at assembly time the address of the pointer to the next FORTH execution address in a colon-definition to be interpreted.

At run-time, NEXT moves IP ahead within a colon-definition. Therefore, IP points just after the execution address being interpreted. If an in-line data structure has been compiled (i.e., a character string), indexing ahead by IP can access this data.

IP STA, or IP Y LDA,
loads the third byte ahead in the colon-definition being interpreted.

M/CPU n1 n2 -- (compiling assembler)
An assembler defining word used to create assembler mnemonics that have multiple address modes:

1c6e 60 M/CU ADC,
M/CPU creates the word ADC, with two parameters. When ADC, later executes, it uses these parameters, along with stack values and the contents of MODE to calculate and assemble the correct op-code and operand.

MEM Used within the assembler to set MODE to the default value for direct memory addressing, z-page.

MODE -- addr
A variable used within the assembler, which holds a flag indicating the addressing mode of the op-code being generated.

N -- addr (assembling)
Used in a code definition in the form:

N 1 - STA, or N 2+ )Y ADC,
A constant which leaves the address of a 9 byte workspace in z-page. Within a single code definition, free use may be made over the range N-1 thru N+7. See SETUP.

NEXT -- addr (assembling)
A constant which leaves the machine address of the Forth address interpreter. All code definitions must return execution to NEXT, or code that returns to NEXT (i.e., PUSH, PUT, POP, POPTWO).

NOT -- addr (assembling)
When assembling, reverse the condition code for the following conditional. For example:

0= NOT IF, <true part> THEN,
will branch based on 'not equal to zero'.

POPTWO -- addr (assembling)

n1 n2 -- (run-time)
A constant which leaves (during assembly) the machine address of the return point which, at run-time, will pop two 16-bit values from the data stack and continue interpretation.

PUSH -- addr (assembling)
n -- (run-time)
A constant which leaves (during assembly) the machine address of the return point which, at run-time, will pop the 16-bit value from the bottom machine stack byte (as low-byte) to the data stack.

PUT -- addr (assembling)
n1 -- n2 (run-time)
A constant which leaves (during assembly) the machine address of the return point which, at run-time, will write the accumulator (as high-byte) and the bottom machine stack byte (as low-byte) over the existing data stack 16-bit value (n1).

XSAVE -- (run-time)
aux 1 cc -- (assembling)
Occurs in a CODE definition in the form:

BEGIN, . . . cc UNTIL,
At run-time, UNTIL, controls the conditional branching back to BEGIN. If the processor status bit specified by cc is false, execution returns to BEGIN; otherwise execution continues ahead.

At assembly time, UNTIL, assembles a conditional relative branch to addr based on the condition code cc. The number 1 is used for error checking.

UP -- addr (assembling)
Used in a code definition in the form:

UP LDA, or UP )Y STA,
A constant leaving at assembly time the address of the pointer to the base of the user area. i.e.,

HEX 12 # LDY, UP )Y LDA,
loaded the low byte of the third user variable, DP.

W -- addr (assembling)
Used in a code definition in the form:

W 1+ STA, or W 1 - JMP, or W )Y ADC,
A constant which leaves at assembly time the address of the pointer to the code field (execution address) of the Forth dictionary word being executed. Indexing relative to W can yield any byte in the definition's parameter field. i.e.,

2 # LDY, W )Y LDA,
fetched the first byte of the parameter field.

Specify 'indexed indirect X' addressing mode for the next op-code generated.

XSAVE -- addr (assembling)

Used in a code definition in the form:

XSAVE STX, or XSAVE LDX,
A constant which leaves the address at assembly time of a temporary buffer for saving the X register. Since the X register indexes to the data stack in z-page, it must be saved and restored when used for other purposes.
FORTH Assembler for 6502 by W. F. Ragsdale  
July 1, 1980

SCR # 81
0 ( FORTH-65 ASSEMBLER  
   WFR-79JUN03 )
1 HEX
2 VOCABULARY ASSEMBLER IMMEDIATE ASSEMBLER DEFINITIONS
3
4 ( REGISTER ASSIGNMENT SPECIFIC TO IMPLEMENTATION )
5 EQ CONSTANT XSAVE  
6 D9 CONSTANT IF  
7 D1 CONSTANT N
8 ( NUCLEUS LOCATIONS ARE IMPLEMENTATION SPECIFIC )
9 ' (DO) 0E + CONSTANT POP
10 ' (DO) OC + CONSTANT POPTW
11 ' LIT 13 + CONSTANT PUT
12 ' LIT 11 + CONSTANT PUSH
13 ' LIT 18 + CONSTANT NEXT
14 ' EXECUTE MPA 11 - CONSTANT SETUP

SCR # 82
0 (  
   ASSEMBLER, CONT.  
   WFR-780CT03 )
1 0 VARIABLE INDEX -2 ALLOT
2 0909, 1505, 0115, 8011, 8009, 100D, 8019, 8080,
3 0080, 1404, 8014, 8080, 8080, 1C8C, 801C, 2C80,
4 5 2 VARIABLE MODE
6 :A 0 MODE 1;  
7 :X 3 MODE 1;  
8 :Y 6 MODE 1;
9 10 :BOT ,X 0 ;  
11 :SEC ,X 2 ;  
12 :RF) ,X 101 ;
13 14 15

SCR # 83
0 (  
   UPMODE, CPU  
   WFR-780CT23 )
1 2 : UPMODE IF MODE @ 8 AND 0= IF 8 MODE +1 THEN THEN
3 1 MODE @ 0= AND -DUP IF 0 DO DUP + LOOP THEN
4 OVER 1+ @ AND 0= ;
5 6 : CPU <BUILDS C, DOES> C@ C, MEM ;
7 00 CPU BRK,  
8 B8 CPU CLV,  
9 C8 CPU INY,  
10 68 CPU PLA,  
11 38 CPU SEC,  
12 46 CPU TAX,  
13 98 CPU TYA,
14 15

FORTH DIMENSIONS III/5  
Page 149
SCR # 84
0 ( M/CPU, MULTI-MODE OP-CODES WFR-79MAR26 )
1 : M/CPU <BUILDS C, , DOES>
2   DUP I+ @ 80 AND IF 10 MODE +1 THEN OVER
3   FF00 AND UMODE UMODE IF MEM CR LATEST ID.
4   3 ERROR THEN C@ MODE C@ INDEX + C@ + C, MODE C@ 7 AND IF MODE C@ OF AND 7 IF C, ELSE THEN THEN MEM ;
5
6   1C6E 60 M/CPU ADC, 1C6E 20 M/CPU AND, 1C6E CO M/CPU CMP,
7   1C6E 40 M/CPU EOR, 1C6E A0 M/CPU LDA, 1C6E 00 M/CPU ORA,
8   1C6E 80 M/CPU SBC, 1C6C 80 M/CPU STA, 0D0D 01 M/CPU ASL,
9   0D0C CI M/CPU DEC, 0D0C EI M/CPU INC, 0D0D 41 M/CPU LSR,
10  0D0D 21 M/CPU ROL, 0D0D 61 M/CPU ROR, 0414 81 M/CPU STX,
11  0486 00 M/CPU CPI, 0486 CO M/CPU CPY, 1496 A2 M/CPU LDX,
12  048E 00 M/CPU LDY, 048E 80 M/CPU STY, 0480 14 M/CPU JSR,
13  8480 40 M/CPU JMP, 0484 20 M/CPU BIT,

SCR # 85
0 ( ASSEMBLER CONDITIONALS WFR-79MAR26 )
1   BEGIN, HERE I+ ; IMMEDIATE
2   UNTIL, ?EXEC >R i ?PAIRS R> C, HERE I+ - C, ; IMMEDIATE
3   IF, C, HERE O C, 2 ; IMMEDIATE
4   THEN, ?EXEC 2 ?PAIRS HERE OVER C@ IF SWAP ! ELSE OVER 1+ - SWAP C1 THEN ; IMMEDIATE
5   ELSE, 2 ?PAIRS HERE I+ 1 JMP,
6   SWAP HERE OVER 1+ - SWAP C1 2 ; IMMEDIATE
7   : NOT 20 + ; ( REVERSE ASSEMBLY TEST )
8   90 CONSTANT CS ( ASSEMBLE TEST FOR CARRY SET )
9   10 CONSTANT O< ( ASSEMBLE TEST FOR LESS THAN ZERO )
10  90 CONSTANT >= ( ASSEMBLE TEST FOR GREATER OR EQUAL ZERO )
11
12
13
14
15

SCR # 86
0 ( USE OF ASSEMBLER WFR-79APR28 )
1   : END-CODE ( END OF CODE DEFINITION *)
2   CURRENT @ CONTEXT ! ?EXEC ?CSP SMUDGE ; IMMEDIATE
3
4   FORTH DEFINITIONS DECIMAL
5   CODE ( CREATE WORD AT ASSEMBLY CODE LEVEL *)
6   ?EXEC CREATE [COMPILE] ASSEMBLER
7   ASSEMBLER MEM !CSP ; IMMEDIATE
8
9 ( LOCK ASSEMBLER INTO SYSTEM )
10  ' ASSEMBLER CFA ' ;CODE 8 + ! ( OVER-WRITE SMUDGE )
11  LATEST 12 +ORIGIN ! ( TOP NFA )
12  HERE 28 +ORIGIN ! ( FENCE )
13  HERE 30 +ORIGIN ! ( DP )
14  ' ASSEMBLER 6 + 32 +ORIGIN ! ( VOC-LINK )
15  HERE FENCE !
Finally, we defined another 22 bytes, for a total of 24 bytes in the dictionary for the value of DAYS/MONTH. This article is written to illustrate 4 different ways of implementing a simple Table Lookup operation. The goal is the following: we want to create a FORTH word, named DAYS/MONTH which behaves as follows: Given an index on the stack which is the month number, such as 1 for January and 12 for December, we want to return the number of days in that month, in a normal year. Thus if we execute 6 DAYS/MONTH it should return 30, which is the number of days in the month June. I will use the Starting FORTH dialect in this paper, not fig-FORTH, so if you try to type in the examples, they probably won't work unless you are running a system that behaves as described in Starting FORTH (or the 79-Standard).

Our first attempt at solving this problem uses the FORTH word VARIABLE. The code is as follows:

```
VARIABLE DAYS/MONTH 22 ALLOT
31 'DAYS/MONTH 28 'DAYS/MONTH 31 'DAYS/MONTH 30 'DAYS/MONTH 31 'DAYS/MONTH 30 'DAYS/MONTH 31 'DAYS/MONTH 30 'DAYS/MONTH 31 'DAYS/MONTH
```

There is nothing significant about the ' (apostrophe), I only prefaced the VARIABLE name with it because I want to use the word DAYS/MONTH later. Now, what happened is that VARIABLE allocated 2 bytes in the dictionary for the value of DAYS/MONTH. The 22 ALLOT then allocated another 22 bytes, for a total of 24 bytes, or 2*12 cells. We next proceeded to initialize the values that were allocated by explicitly calculating the offsets and storing in the appropriate location. Finally, we defined DAYS/MONTH as a colon definition which performs arithmetic on the index, adds it to the start of the table, and fetches the result. Now, let's look at another way of doing this that requires less typing and is also more general. We will first define a word called TABLE which will aid us in the creation of tables like the one above. What we will do is first place the initial values of the TABLE on the stack, together with the number of the initial values. Then, we will define TABLE to copy these into the dictionary. Here is how it works:

```
: TABLE ( N N-n- ... NI n--- )
  "DO " LOOP ;
CREATE 'DAYS/MONTH
31 30 31 30 31 31 30 31 30 31 28 31 12 TABLE

: DAYS/MONTH ( INDEX --- VALUE )
  1- 2* 'DAYS/MONTH + @ ;
```

Now this is considerably less typing than the first way of doing it, but notice that I had to reverse the order of the days per month since that is the way stacks behave. I used CREATE instead of VARIABLE because it does not allocate any space for the initial value, but otherwise behaves just like VARIABLE. The access word DAYS/MONTH is identical to before.

I am still not satisfied, however, so let's try it yet another way. Instead of defining TABLE to add values to the dictionary with , (comma) why not just use , directly?

```
CREATE 'DAYS/MONTH
31 28 31 30 31 30 31 30 31 30 31 28

: DAYS/MONTH ( INDEX --- VALUE )
  1- 2* 'DAYS/MONTH + @ ;
```

Now we are getting somewhere! If we simply use the FORTH word , (comma) to add the value to the dictionary, see how simple and readable it becomes. The values are just typed in and separated by commas! Is it possible to improve even on this? FORT14 should show you how.

```
Note that the above examples illustrate the immense diversity available in FORTH. There is no obvious right or wrong, and the simplest and usually most general solution to a given problem must be discovered, usually by trial and error. FORTH's biggest virtue, in my opinion, is that it makes the trial and error process extremely efficient, and therefore, allows people to experiment and discover the best solution for themselves.
```

```
HELP WANTED

Programmers needed to produce new polyFORTH systems and applications. Two to three years experience working with mini/micro computers and peripherals.

Contact: Patricia Jones
FORTH, INC.
2309 Pacific Coast Highway
Hermosa Beach, CA 90254
(213) 397-8493
```

```
fig-FORTH NOVA GROUP

Mr. Francis Saint, 2218 Lulu, Wichita, KS 67211, (316) 261-6280 (days) has formed a FIG Group to trade information and assistance between fig-FORTH NOVA users.

Pub. Comment: Hope to see a new, clean listing. How about some other specific groups?
```
THE GAME OF REVERSE
M. Burton

REVERSE is a number game written in FORTH, primarily as an exercise in array manipulation. The object of REVERSE is to arrange a list of numbers (1 through 9) in ascending numerical order from left to right. Moves are made by reversing a subset of the list (from the left). For example, if the current list is 2 3 4 5 1 6 7 8 9 and four numbers are reversed, the list will be 2 3 4 1 5 7 6 9 8 then if five numbers are reversed, the game is won.

1 2 3 4 5 6 8 9

To leave a game that is in progress, simply reverse zero numbers.

REVERSE Glossary

SEED
The number seed for the pseudorandom number generator. SEED is initialized as the REVERSE words are compiled, by hitting any key on the console.

MOVES
Keeps track of the number of moves made in a REVERSE game. If more than fifteen moves are made to win, you haven't got the hang of the game.

RND
The pseudorandom number generator, courtesy of FORTH DIMENSIONS. RND generates random number in the range 0 through range-1. RND is used to scramble the number list.

DIM
A defining word used in the form
DIM n

Produces an n+1 length word array named xxxx, with elements 0 through n. For the REVERSE application, element 0 is not used.

Y/N
-- flag
Solicits an input string from the console, then checks the first character of the string for an uppercase or lowercase

SCR # 228
0 ( The Game of Reverse (SEED, MOVES, RND, DIM, Y/N) 101281-MPB )
1 2 0 VARIABLE SEED ( Seed for random number generator )
3 0 VARIABLE MOVES ( Number of reverses so far )
4 CR " Please depress any key:" ( Fertilize the seed )
5 KEY SEED 1
6 7 : RND ( Random number generator range -- rnd# )
8 SEED @ 259 * 3 + 32767 AND DUP SEED 1 32767 */ ;
9 10 : DIM ( Reserve an integer word array n -- )
11 <BUILDS 1+ 2 * ALLOT
12 DOES> ;
13 14 : Y/N ( Get a Y or N response -- flag )
15 PAD 80 EXPECT PAD C@ CR CR 95 AND B9 = ; -->

SCR # 229
0 ( The Game of Reverse (Game instructions) 101281-MPB )
1 2 : INSTRUCT CR CR 18 SPACES "The Game of REVERSE"
3 CR CR ." Would you like instructions? " Y/N
4 IF ." The object of the game is to arrange a random list"
5 CR ." of nine numbers into ascending numerical order in"
6 CR ." as few moves as possible by reversing a subset of"
7 CR ." the list. For example, given the random list," CR
8 CR ." reversing a subset of 4 would yield the list," CR
9 CR ." 8 4 2 5 7 3 9 1 6 " CR
10 CR ." To quit the game, simply reverse 0." CR CR
11 THEN
12 13 -->
14

SCR # 230
0 ( The Game of Reverse (ARRAY operations) 101281-MPB )
1 2 9 DIM ARRAY ( Reserve a ten word array )
3 4 : A@ ( Fetch an array word index -- array.value )
5 2 * ARRAY @ + 0 ;
6 7 : A! ( Store an array word array.value\index -- )
8 2 * ARRAY + 1 ;
9 10 : AINIT ( Initialize ARRAY -- )
11 10 1 DO I DUP A@ LOOP ;
12
13 : A. ( Print ARRAY -- )
14 CR ." The list is now..." CR
15 CR 6 SPACES 10 1 DO I A@ 3 .R LOOP ; -->

fig-FORTH Version 1.15 M. Burton
**SCR # 231**

0 ( The Game of Reverse [ARRAY operations, cont.] 100781-MPB )

1 2: ASCRAMBLE ( Mix up the array values -- )

3 1 9 DO I RND 1+ ( Calculate K )

4 I A# ( Get ARRAY[I] value )

5 OVER A@ ( Get ARRAY[K] value )

6 I A! ( Store ARRAY[K] in ARRAY[I] )

7 SWAP A! -1 ( Store ARRAY[I] in ARRAY[K] )

8 +LOOP ;

9

10 : GETIN ( Get amount to reverse -- n )

11 BEGIN CR " Reverse how many? "

12 PAD 00 EXPECT PAD 1 48 -

13 DUP 0 < OVER 9 > OR DUP

14 IF CR " Only 0 through 9 is allowed. " THEN 0=

15 UNTIL CR -->

**SCR # 232**

0 ( The Game of Reverse [ARRAY operations, cont.] 100781-MPB )

1 2: AREVERSE ( Reverse a subset of ARRAY n -- )

3 DUP 2 / 1+ 1 ( Loop limits are 1 to (n/2)+1 )

4 DO DUP I - 1+ ( Calculate index n-I+1 )

5 DUP A# SWAP ( Get ARRAY[n-I+1] )

6 I A# ( Get ARRAY[I] )

7 SWAP A! ( Store ARRAY[I] in ARRAY[n-I+1] )

8 I A! ( Store ARRAY[n-I+1] in ARRAY[I] )

9 LOOP DROP ;

10 : ACHECK ( Check for ascending seq. -- flag )

11 1 10 1 DO

12 I DUP A# = AND

13 LOOP ;

15 -->

**SCR # 233**

0 ( The Game of Reverse [REVERSE definition] 101281-MPB )

1 2: REVERSE ( Play the game )

3 INSTRUCT AINIT BEGIN

4 ASCRAMBLE 0 MOVES ! BEGIN

5 A. GETIN DUP 0=

6 IF 1 ELSE

7 AREVERSE 1 MOVES +! ACHECK

8 THEN

9 UNTIL A. CR " You made " MOVES @ " reversals. " CR

10 UNTIL CR " Care to play again? " Y/N 0=

11 UNTIL CR " Thanks for playing REVERSE... " CR CR ;

`fig-FORTH Version 1.15  M. Burton`

ok
THE 31 GAME
Written by Tony Lewis 11/81

The "31 Game" is an attempt to use FORTH fundamentals to produce an entertaining result. The object is to entice you into analyzing both the game itself and the methods used to produce it. The game buffs might wish to know that I have been an avid "player" (not gambler) for over 30 years and have made extensive practical studies of various games. Any phone communication is welcome. I am two years behind in my written correspondence; so sending me letters which require replies will prove futile. The program is my first effort in FORTH. However, I have had extensive experience with six different main frame assemblers plus a little COBOL of the late 60's vintage. Any constructive suggestions on general style and technique are welcome, but I am not really interested in being told that I could have shaved 100 microseconds from my run time or saved fifteen bytases of memory. In fact, there are indeed extraneous "Cl's" which were included to get good hard copy, also.

This program was written in micro-motion (c) FORTH-79 Version 1.2 to be run on a 48K Apple II.

Therefore, the following words are non-standard but included in the micro-motion FORTH.

- HOME position the cursor to the upper left corner of the CRT and clear the CRT to blanks.

- CV and CH are used to position the input cursor anywhere on the text window per Ex. 4 CV 10 CH moves the cursor to the 4th (pun) row 10th column of screen.

- SETINV, SETFLASH, and SETNORM set flags in the Apple output subroutines which respectively cause all subsequent characters to be displayed on the text screen inverse, flashing and normal mode without affecting characters already displayed.

- In closing, I wish to thank Bill Ragsdale for his gracious support and I especially acknowledge the incredibly patient treatment I received from Phil Wasson of Micronomy as he neatly led me through my FORTH initiation.

Tony Lewis
109 Marinier Dr.
Corte Madera, CA 94925
(415) 924-1481
(415) 924-4216 (late hours)

*Apple is a registered trademark of Apple Computer, Inc.
SCREEN4
( PLAY THE GAME OF 31 BY TONY LEWIS )
: MAIN1 ( LOGIC SHELL ) O GAMEWIN !
: BEGIN
" NEW GAME? "; TYPE Y OR N OR ( BIG BET )," OR OR KEYS UP TO 70 = I CHECK FOR N 
( FALSE LEAVES 78 ON STACK FOR 'UNTIL' )
IF NEWGAME = "Y" THEN
IF GAMESWIN 2 = 1 THEN
" YOU MUST WIN 3 GAMES TO " . ; " PLAY 'BIG BET' "
THEN
ELSE NORMAL1
THEN
UNTIL 1:
: NEWGAME HOWDOSI CODECONS MAIN51 ;
CODECONS MAIN51
NEWGAME: TYPE Y OR N OR ( BIG BET ).
DO YOU WANT FIRST PLAY? TYPE Y OR N, 
MY PLAY IS 0
THE DECK NOW CONTAINS
1 1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
THE RUNNING TOTAL IS 2
TYPE IN CARD 1 = 6
YOUR CARD WAS A 3
THE DECK NOW CONTAINS
1 1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
THE RUNNING TOTAL IS 10
TYPE IN CARD 1 = 6
YOUR CARD WAS A 2
THE DECK NOW CONTAINS
1 1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
THE RUNNING TOTAL IS 16
MY PLAY IS 1
THE DECK NOW CONTAINS
1 1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
THE RUNNING TOTAL IS 18
MY PLAY IS 1
THE DECK NOW CONTAINS
1 1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
THE RUNNING TOTAL IS 17
TYPE IN CARD 1 = 6
YOUR CARD WAS A 3
THE DECK NOW CONTAINS
1 1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
THE RUNNING TOTAL IS 20

FORTH CLASSES

LAXEN AND HARRIS, INC.
24301 Southland Drive
Hayward, CA 94545
(415) 887-2894
Introductory classes
Process control
Applications programming
Systems level programming

GREG STEVENSON
Anaheim, CA
(714) 523-4202
Introductory classes

KNOWARE INSTITUTE OF TECHNOLOGY
Box 8222
Stanford, CA 94305
(408) 338-2720
Introductory classes
Graphics classes

INNER ACCESS CORPORATION
Belmont, CA
(415) 591-8295
Introductory classes

FORTH, INC.
2309 Pacific Coast Highway
Hermosa Beach, CA 90254
(213) 372-8493
Introductory classes
Advanced classes

FORTH DIMENSIONS III/5
In this article, I am going to tell a true story. For those people who think FORTH is a religion, they might just consider this to be my testimony.

Last November, I had access to a very little known, but well built microcomputer -- MX 964/2 by Columbia Data Products, Inc. of Maryland. This little machine has two 8080 CPUs. One is for the Host and the other for terminal. There are 64 K of RAM in the Host, and 32 K of the Terminal RAM is dedicated to the 512 x 256 bit mapped graphics. It also includes a 9" CRT, 2 double density drives, keyboard, 4 serial ports, and 4 parallel ports. Its all in one piece. It boots up with whatever operating system is on the disk after powered up and the carriage return key has been pressed. Beautiful isn't it?

However, there is a big problem, as with most microcomputer companies, the instruction manuals are terrible. And I mean terrible! Let me just give you one example: "For this information please see figure ...", only to find there was no such figure and no page number.

Graphics are one of the most important features with this machine. 512 x 256 bit mapped graphics is the best that can be expected under the price allowance. There are quite a few well known microcomputers on the market claiming High Resolution Graphics. But those High Resolution ones are just like a big blob compared with the individual pixel that bit mapped. So, I have a nice machine with all the fancy graphic capabilities, but lacking the key to open it. Anxiety mounts up quickly.

I have a friend who's an excellent 8080/80 assembly programmer. He implemented UCSD Pascal for a microcomputer. Naturally, since he was the first one, it seemed logical to seek his help. With a poorly written computer manual...
I at least knew that the graphic part simulates Tektronics 4010, so he spent a whole week just trying to draw one more square along the edges of the CRT. Seemingly it would be an easy job, but even so it never came near to what he would have liked. Later on, I spent a couple of weeks twiddling with Microsoft BASIC compiler and it also produced lousy results.

At the same time, I received my 8080 fig-FORTH listing. So, I typed the whole 60 K of assembly listing with the lousiest text editor (i.e., ED.COM). It was a monumental job. Nevertheless, I had the fig-FORTH up and running.

By now, I was very desperate to get it going. Equipped with the FORTH power and the poor manual, I set forth to try the graphics again. Again, I sought help from a friend who works for Tektronics and is experienced with FORTH. With FORTH, the whole task turned into a very simple job, compared to the previous attempts we had with the assembly and BASIC. Thus, now I am steadfast in my belief in FORTH.

Screen 10 and 11 sets up the variables and the Columbia Mx96 hardware dependent words. The X-coordinate starts at the lower left corner as 0, far right as 1023, while Y = 0 starts at the lower left corner to the top as 779. Screen 12 to 14 defines the basic words, which draw the line, move the cursor, relative draw and move. Screen 15 defines the words to draw a square and the erasing of it. Screen 16 lets me draw many squares.

I know that there are still a lot of nice words that can be written, such as, to draw triangles, curve lines, etc. But, from this small exercise, I am totally convinced the FORTH is the one I will use from now on.
A VIDEO VERSION OF MASTER MIND

David Butler
Dorado Systems

The writing of this program served as my introduction to FORTH. Using the FORTH Installation Manual, I stumbled my way through the basic concepts of FORTH and eventually arrived at this video Master Mind game. The game is derived completely from the original board version of Master Mind, therefore, all credit for the game itself goes to the Invicta Game Company.

The program contains many of the functions found in video editors, including cursor management and character collection. The sequence of this computer version of the game is as follows: After displaying the directions, the program prompts the player to enter his skill level. Then a 'secret code' is generated with the help of the player tapping the space bar. The screen is cleared, and a 'mask' of the Master Mind playing board is displayed. The cursor lands at the location where the player is to begin entering his guess. The program retains control of the cursor, responding to the player's key strokes. Backspacing and tabbing are allowed, en-

SCR # 18
0 ( Master Mind in FORTH by David A. Butler DAB-17nov80 )
1 -->
2 David A. Butler
3 33300 Mission Blvd
4 Apt 126
5 Union City, CA. 94587
6 (415) 487-6039
7
8
9 **** A note about style: If there is any, it is an accident.
10 This was my first application in FORTH, so it may lack
11 some elegance.
12
13 **** Requirements: A video display 80 x 24 characters,
14 cursor addressing and clear screen functions.

SCR # 19
0 ( Master Mind -notes- DAB-17nov80 )
1 -->
2 This is an implementation of Master Mind by Invicta.
3 The game is very popular because it is easy to learn and a
4 challenge to play. There is a bit of luck to it, but it is
5 mainly an exercise in logical deduction. A "secret" code is
6 generated, and it is "cracked" by analyzing a set of clues.
7
8 Those familiar with the original board game will have no
9 difficulty adjusting to the computer version. To newcomers,
10 follow the directions carefully and you will have it in no
11 time. The Forth version is functionally identical to the
12 board version. It is written in f-Forth, and has been run
13 successfully on 6502, 8090, Z80, and 68000 processors. It
14 is a good demonstration program as well as an enjoyable game.
15
SCR # 20
0 ( Master Mind set up some variables DAB-17nov80 )
1
2 : TASK 1 ( FORGETTABLE MARKER )
3 4 VARIABLE COLORS 28 ALLOTT COLORS 30 BLANKS
5 6 VARIABLE CODE 2 ALLOTT 0 VARIABLE GUESS 2 ALLOTT
6 7 VARIABLE SECRET 2 ALLOTT
8 8 VARIABLE BLACKER 0 VARIABLE WHITE
9 9 VARIABLE COLORS
10
11 3 VARIABLE CUR_ROW 25 VARIABLE CUR_COL
12 1 VARIABLE XLOC 1 VARIABLE YLOC 0 VARIABLE DONE
13 -->
14
15
SCR # 21
0 ( Master Mind set up - cont. DAB-17nov80 )
1
2 : C.CONSTANT ," YELLOWRED BLACK GREEN WHITE BLUE "
3 4 VARIABLE COLOR.KEY 6 ALLOTT ( "colors" table )
5 6 ( Use the sum of the ASCII code of the first 3 letters )
7 ( i.e. BLUE = "B" + "L" + "U" = 66 + 76 + 85 = 227 )
8 9 224 COLOR.KEY C' 219 COLOR.KEY 1+ C'
10 207 COLOR.KEY 2+ C' 222 COLOR.KEY 3+ C'
11 232 COLOR.KEY 4+ C' 227 COLOR.KEY 5+ C'
12 96 COLOR.KEY 6+ C'
13
14 0 VARIABLE #ATTEMPTS ( used to keep score )
15 -->
SCR # 22
0 ( Master Mind prompt and randomize DAB-17nov80 )
1 ( These definitions set the random values for the same )
2
3 : NEWCOUNT ( [COLOR# + 1] ) DUP #COLORS 2 <
4 IF 1+ ELSE DROP 1 THEN : 
5
6 : RAND 1 BEGIN NEWCOUNT ?TERMIMAL UNTIL KEY DROP ;
7
8 : ASK,FOR,RANDOM ." To randomize, tap space bar 4 times."
9 0 DO RAND 1 #COLORS + C! LOOP CR ;
10
11 : ASK,FOR,LEVEL
12 CR ." Level 1 or 2 ? " KEY DUP EMIT KEY EMIT
13 50 = IF 7 #COLORS ! ELSE 6 #COLORS ! THEN CR ;
14
15 -->

SCR # 23
0 ( Master Mind translate color to numeric DAB-17nov80 )
1 :
2 : COLOR,FIND ( [COLOR#] -- [J] TYPES COLOR FROM #)
3 1 - 6 * C,CONSTANT 3 + 6 TYPE 1
4
5 : TRANSLATE,CODE
6 ( converts color # from SCODE to COLOR,KEY )
7 ( numeric value in array "SECRET"
8 0 DO SCODE 1 - C, OR 1 - COLOR,KEY + C R : SECRET
9 1 + C! LOOP R ;
10
11
12
13
14 : J R> R> R> [COMPILE] R SWAP OR SWAP OR SWAP OR ;
15 -->

SCR # 24
0 ( Master Mind cursor motion DAB-17nov80 )
1 ( Of course, CRT dependent. Here is Heath )
2 ( *** start CRT dependent words *** )
3 : CURSOR ( [Y] [X] -- [J] ABSOLUTE CURSOR POSITION )
4 31 + SWAP 31 + 89 27 EMIT EMIT EMIT EMIT : 
5
6 : CLEAR ( CLEAR CRT SCREEN ) 27 EMIT 69 EMIT ;
7
8 : HOME ( PUT CURSOR AT HOME POSITION ) 0 0 CURSOR : 
9 ( *** end of CRT dependent words *** )
10 : SHOW.COLORS ( DISPLAY COLOR CHOICES )
11 1 1 DO 1 2 + 58 CURSOR 1 COLOR,FIND LOOP
12 #COLORS 2 = IF 59 57 CURSOR . "<BLANK>" ELSE THEN
13 12 58 CURSOR . "TAB between colors."
14 13 58 CURSOR . "RETURN to get clues."
15 -->

SCR # 25
0 ( Master Mind board layout mask DAB-17nov80 )
1
2 : BAR , !"1" : DASH , " " : ( BOARD SYMBOLS )
3 : TITLE 21 SPACES
4
5 : M A S T E R M I N D ==" "
6 : D A S H E R 2 21 CURSOR BAR 32 0 DO ." " LOOP BAR CR : 
7
8 : SPACER 21 CURSOR ." : ^---------------^---------------^---------------^ : 
9 : CBLOCK DUP CLINE 1+ SPACER 1
10 : HIDDEN 2 23 CURSOR ." XXXX XXXX XXXX XXXX XXXX " : 
11
12 : DISPLAY,BOARD
13 : CLEAR TITLE DASHER HIDDEN 24 3 DO I CBLOCK 2 +LOOP
14
15 -->

Further notes and comments may be found in the source screens.

When the player signals he is ready, the program compares the player's guess to the secret code which was stored away earlier. Clues are generated and displayed, indicating to the player how close he is to the solution. The player has ten chances to deduce the secret code.

There are many improvements which could be made to this program to make advantage of more of FORTH'S built-in vocabulary -- most notably PAD and related words. For those short of memory, note that the directions could be shortened, left out, or read from disk with no change to the overall logic of the program.

FORTH DIMENSIONS III/5 Page 159
SCR # 26
0 ( Master Mind cursor tracking definitions DAB-17nov80 )
1
2: X XLOC @ : Y YLOC @ ;
3
4: XBUMP X 52 =
5 IF 23 DUP CUR.COL ! XLOC 1
6 ELSE 1 XLOC + Y CUR.COL @ 8 + =
7 IF X CUR.COL THEN
8 THEN 1
9
10: UNBUMPX X 23 = IF 52 XLOC ! ELSE 1 XLOC + THEN 1
11
12: TAB CUR.COL @ 47 =
13 IF 23 CUR.COL !
14 ELSE 8 CUR.COL + :;
15 THEN CUR.COL @ XLOC ! DROP Y X CURSOR ; -->

SCR # 27
0 ( Master Mind character collection/editing DAB-17nov80 )
1: BACKSPACE X CUR.COL @ =
2 IF DROP
3 ELSE UNBUMPX Y X CURSOR SPACE Y X CURSOR DROP
4 32 COLORS X 23 + C! 5 THEN 1
6 7: PROCESS ( [CHAR] -- ) PROCESSES CHAR, MANAGES CURSOR
8 DUP EMIT COLORS X 23 + C! XBUMP Y X CURSOR 1
9
10: GET.CHAR KEY DUP 127 =
11 IF BACKSPACE ELSE DUP 9 =
12 IF TAB ELSE DUP 13 =
13 IF 1 DONE ! DROP
14 ELSE PROCESS THEN THEN THEN 1
15 -->

SCR # 28
0 ( Master Mind guess / row section DAB-17nov80 )
1
2
3
4: INITIAL 2 3 3 + DUP YLOC ! CUR.ROW ! 23 23 XLOC
5 CUR.COL ! Y X CURSOR 6
7 30 0 DO 32 1 COLORS + C! LOOP 1
8 9: GET.COLORS INITIAL 0 DONE ! BEGIN GET.CHAR DONE @ UNTIL 1
10
11 PARSE.GUESS 4 0 DO I 8 * COLORS + C! 12
13 18 * COLORS I + C! 14
15 + + I GUESS + C! LOOP 1
16 -->
17
SCR # 29
0 ( Master Mind Clue generation DAB-17nov80 )
1
2: CLUE.CHECK
3 0 BLACKER 1 0 WHITER 1 ( INITIALIZE COUNTS )
4 4 0 DO
5 SECRET I + C! GUESS I + C! = ( CHECK FOR DIRECT HIT )
6 IF 1 BLACKER + 0 1 GUESS + C! 7 THEN LOOP
8 4 0 DO GUESS I + C! 0 IF 1 IF NO HIT 9
4 0 DO
10 GUESS I + C! SECRET J + C! = ( CHECK FOR WHITE )
11 IF 1 WHITER + 1 1 GUESS + C! LEAVE
12 THEN
13 LOOP THEN
14 LOOP 1
15 -->

FORTH DIMENSIONS III/5
SCR # 30
0 ( Master Mind present clues DAB-17nov80 )
1 2: GIVE,CLUES PARSE, GUESS CLUE, CHECK
3 Y 1 CURSOR BLACKER @ : " BLACK "
4 WHITER @ : " WHITE "
5 : UNMASK 3 23 CURSOR
6 4 0 DO 1 SCOEE + @ COLOR,FIND 1 3 =
7 IF, " ELSE, " THEN LOOP 23 1 CURSOR ;
8 9 : ?AGAIN 20 58 CURSOR , " TYPE MASTER TO" 21 58 CURSOR
10 : " PLAY AGAIN." UNMASK 23 1 CURSOR ;
11 : LOSER 16 58 CURSOR , " NICE TRY BUT" 17 58 CURSOR
12 : " NO CIGAR." ?AGAIN ;
13 : WINNER 16 58 CURSOR , " PRECISELY." #ATTEMPTS ?
14 , " TRYS." ?AGAIN ;
15 -->

SCR # 31
0 ( Master Mind Directions to player DAB-17nov80 )
1 2: DIRECTIONS CLEAR CR CR CR CR CR
3 10 0 DO LOOP , " Welcome to MASTER MIND." CR CR
4 , " The object of Master Mind is to break the secret code." CR CR
5 CR , " The computer will pick the secret code, and you must"
6 CR , " figure it out. Two kinds of clues are given:" CR CR
7 CR , " (1) # BLACK means that you have # miss correct " CR CR
8 CR , " in both color and position." CR CR
9 CR , " (2) # WHITE means that you have # miss of the " CR CR
10 , " correct color that are incorrectly" CR CR
11 placed. " CR CR
12 , " Be sure to spell the colors correctly. You may tab amounts " CR CR
13 CR , " the 4 positions until you've make your best guess." CR CR
14 , " Type [RETURN] to receive clues." CR CR , " Good luck." CR
15 CR CR 1 -->

SCR # 32
0 ( Master Mind ++ FINAL ++ DAB-17nov80 )
1 2:
3 2: MASTER DIRECTIONS 0 0 #ATTEMPTS !
4 ASK,FOR,LEVEL ASK,FOR,RANDOM
5 DISPLAY,BOARD TRANSLATE,CLUE ( put UNMASK to debug )
6 0 10 DO
7 1 #ATTEMPTS +=
8 PARSE,GUESS 1 GET.COLORS GIVE,CLUES
9 BLACKER @ = 4
10 IF WINNER LEAVE
11 ELSE THEN
12 -1 #LOOP
13 BLACKER @ < 4
14 IF LOSER ELSE THEN 1: MASTER
15

(There are hidden colors)

 denen

1 RED RED BLUE GREEN
2 BLUE RED BLUE GREEN
3 RED RED BLUE GREEN
4 BLACK 0 WHITE RED RED BLUE GREEN
5 BLACK 0 RED RED BLUE GREEN
6 RED 0 WHITE RED RED BLUE GREEN
7 BLUE 0 WHITE RED RED BLUE GREEN
8 RED 0 WHITE RED RED BLUE GREEN
9 BLUE 0 WHITE RED RED BLUE GREEN
10 RED 0 WHITE RED RED BLUE GREEN
11 BLUE 0 WHITE RED RED BLUE GREEN

12 BLACK 0 WHITE RED RED BLUE GREEN
13 BLACK 0 RED RED BLUE GREEN
14 BLACK 0 WHITE RED RED BLUE GREEN
15 BLACK 0 WHITE RED RED BLUE GREEN

16 BLACK 0 WHITE RED RED BLUE GREEN
17 BLACK 0 RED RED BLUE GREEN
18 BLACK 0 WHITE RED RED BLUE GREEN
19 BLACK 0 WHITE RED RED BLUE GREEN
20 BLACK 0 WHITE RED RED BLUE GREEN

1: Snapshot of board after playing Master Mind )

ANNOUNCEMENTS

NEW JERSEY FIG CHAPTER BEING FORMED

Interested parties should contact:
George B. Lyons
280 Henderson St.
(212) 696-7606 - days
(201) 451-2905 - eve

BOSTON FIG CHAPTER SEEKING MEMBERS

Interested parties should contact:
R. I. Demrow
P. O. Box 158, Blv. Sta.
Andover, MA 01810
(617) 389-6400 x 190 - work
(617) 644-5796 - home

MOUNTAIN WEST FIG CHAPTER ORGANIZING

Interested parties in the greater Salt Lake City area should contact:
Bill Haywood
(801) 962-8000

TECHNICAL PRODUCTS CO. MOVES

New address:
P. O. Box 2358
Boone, NC 28607-2358

FIG NEW YORK CITY MEETING CONTACT

James Basile
40 Circle Drive
Westbury, NY 11590
(516) 333-1298

DALLAS-FT. WORTH METROPLEX FIG MEETING CHANGE

Meetings now being held at:
Software Automation, Inc.
1003 Business Parkway
Richardson, TX
contact:
Marvin Elder (214) 231-9142
Bill Drisell (214) 264-9680

FORTH DIMENSIONS III/5 Page 161
TRANSFER OF FORTH SCREENS BY MODEM

Guy T. Grotke
Forth Gear
San Diego, CA

Here is a simple but hopefully useful set of definitions for serial transfer of FORTH screens between machines. Several of us in the San Diego FIC are interested in sharing software, but we have been unable to do so because of all the different disk formats in use. While only a few had access to similar machines, we took a poll and found that more than 90% had RS-232 ports. The following two screens permit unidirectional transfer with a modem over telephone lines at 300 baud or hardwired at 19200 baud. The definitions are not particularly sophisticated. There is no error checking or ack/nack with retry. Since it is source code which is being transferred, some editing will probably be necessary anyway, so such safeguards aren't worth the effort to write them.

There are four definitions which are entirely system dependent in each screen. These are SOUTPUT, COUTPUT, SINPUT, and CINPUT. Respectively, they direct output to the serial port, output to the console, input from the serial port, and input from the console. If your system doesn't use I/O flags or vectors, you may have to write serial port drivers and point KEY and EMIT to them for SOUTPUT and SINPUT. In screen 80, these four words are defined for an APPLE running a serial interface in slot two (driver at $C200). In screen 58, they are defined for an Ohio Scientific with the normal serial port found in the personal models. These are examples of vectored and flagged I/O redirection.

The remaining definitions should be quite universal among fig (and other) systems. Screen 80 contains all that is necessary to receive screens under the control of the sender. FINISHED and RECEIVE simply redirect input and output. The word P redefines the fig editor word P to do the same thing except with I/O redirection. Note that these three definitions are simple and fool-proof enough that they could be sent to another computer if that computer was first told to accept all input from the serial transfer channel. Once these definitions were compiled by the receiving system, screen transfer could begin. In screen 58, the word WAIT waits for anything to be sent back from the receiver with a carriage return on the end. The word OK is defined just in case the receiver sends one or more OK's back to the sender during transfer. SEND-SCREEN will send a screen to the receiver, one line at a time, by emulating a user entering lines with the receiver's line editor. First SEND-SCREEN asks the receiver to list the screen being sent.

This insures that the proper disk blocks are resident. After the LIST, the receiver will reply "OK" followed by a carriage return. WAIT makes the transmitter wait for this carriage return. This is the only handshaking needed. Each line of text is sent preceded with the letter P and a space, and followed by a carriage return. WAIT causes the transmitter to wait for the receiver to reply "OK" after each line is sent. SEND is a multi-screen transmitter. Note that the range of screens received and recorded on disk will correspond exactly to the screen numbers sent.

HELP WANTED

Part-time - New York-New Jersey Area

Assist internationally known sound artist, Max Neuhaus, develop additional software for microcomputer controlled sound synthesis system. FORTH controlling 32 synthesizers from CRT Light Pen Terminal.

Moderate fees, travel possibilities, hardware experience preferred. Send information or resume to:
Max Neuhaus
210 5th Avenue
New York, NY 10010

Independent FORTH programmers to implement FORTH for RS-232, Apple, CP/M and other systems. Royalties paid for best implementation with most enhancements. Great opportunity for the competitive programmer who, like me, would like to make a living at home and not have to move to California to do it.

Contact:
Marc Perkel
Perkel Software Systems
1636 N. Sherman
Springfield, MO 65803
(417) 862-9030

If that is inconvenient, a variable containing an offset or starting receiver screen number could be added.

The proof that it works is before you: the different screen formats and distant screen numbers reflect the fact that screen 58 was written on my OSI and sent to my APPLE to be printed. I have used these definitions to send a 6502 assembler, a database manager, and several hundred data entries between my machines with no trouble.
PRODUCTS REVIEW

SORCERER-FORTH
by Quality Software

For about a year, I have been using an excellent version of fig-FORTH tailored for the Exidy Sorcerer. It is a product of Quality Software, 6660 Reseda Blvd., Suite 105, Reseda, CA 91335.

FORTH for the Sorcerer implements Release 1.1 of 6808 fig-FORTH. It includes a full screen editor and input/output routines for the keyboard, screen, and both serial and Centronics printers. The Sorcerer's excellent graphics are also available.

Disc storage is simulated in RAM. A 32 K Sorcerer can hold 14 screens—with 48 K, up to 30 screens. Tape-handling routines are provided, to move data to and from the simulated disk space. The CP/M disk interface routines are present, but not implemented.

One of the nicest features of Quality Software's FORTh is its documentation. The 126-page manual is well-written, and relatively complete. It includes sufficient information for a FORTH neophyte, though it does not delve too deeply into system operations.

Quality Software permits—even encourages—users to market application programs incorporating Sorcerer FORTH. They do ask that written permission be obtained first, but promise that permission will normally be granted after review of a sample of the program.

I highly recommend this excellent product, and ask that you include it in your periodic listing of available software.

C. Kevin McCabe
1560 N. Sandburg Terr. #4105
Chicago, IL 60610
(312) 664-1632

A COMPARISON OF TRANSFORTH WITH FORTH
Insoft
Medford, OR

A question we've been hearing a lot lately is "How does TransFORTH compare with fig-FORTH?" In structure, TransFORTH is similar to most versions of FORTh but is not a FORTh-79 Standard System. The major differences are outlined in this paper.

Floating-point numbers

In TransFORTH, the stack itself contains floating-point numbers, with 9 digits of accuracy. No special sequences are required to retrieve floating-point values. Words are available for storing or retrieving single bytes and two-byte cells, but all values are stored on the stack in floating-point format. Numbers can be as large as 1E38, and as small as 1E-38.

Transcendental functions

The floating-point format mentioned above makes TransFORTH a natural language for transcendental functions. Functions included in the system which are not found in most versions of FORTh includes: cosine, tangent, square root, and powers.

Data structures

TransFORTH contains words that will store or fetch 5-byte floating-point values, 2-byte cells, and single bytes from any location in memory. TransFORTH does not have the fig-FORTH <BUILDS, DOES> construction, but instead uses a powerful built-in array declaration. Arrays can either fill space in the dictionary, or be located absolutely in memory. Arrays with any number of dimensions may be declared, and each dimension can have any length, within the limits of available memory.

Strings

Strings are merely arrays (of any dimension and size) with an element length of one. Each character occupies one byte, i.e., one element of the array. Built-in string functions included.

Disk access and the editor

TransFORTH does not use the virtual memory arrangement found in most versions of FORTh. Instead a standard DOS 3.3 format is used, and files are called from the disk by name.

TransFORTH includes a straightforward line-based text editor. The editor is not added to the dictionary as a list of defined words, but is included as a separate module callable from TransFORTH. DOS text files are used for saving source files. This means that any text editor that uses DOS text files may be used for editing TransFORTH programs. In addition, TransFORTH program data may be shared with other programs and languages.

Graphics

Two graphics utilities along with a couple of graphics demo programs are included on the system diskette. One utility contains high-resolution graphics and Turtlegraphics commands, and the other has low resolution graphics commands. The graphics capabilities are added to the system by compiling these utilities into the dictionary. The hi-res package includes a call to a module which allows text and graphics to appear together anywhere on the screen.

Vocabulary

TransFORTH is a single-vocabulary system. Related programs can be grouped together in disk files, rather than in separate vocabularies. (Multiple vocabularies find their most usage in multi-user systems.)

Compilation and speed

All entries in TransFORTH are compiled directly into 6902 machine language for greater speed. No address interpreter is used. Even immediate keyboard entries are compiled before being executed. This means that routines can be tested at the keyboard for speed before being added as colon definitions.

TransFORTH is fast. It is not as fast as an integer versions of FORTh, because it handles 5 bytes with every stack manipulation instead of two. TransFORTH programs will run faster than similar Apple-soft programs, and show a great increase in speed when longer programs are compared.

While TransFORTH works much like Fig-FORTH, the differences between the two become readily apparent under closer examination. FORTh programmers will pick up TransFORTH with little trouble, but nearly all FORTh programs will require translation into TransFORTH to take advantage of its powerful features. These features are accessible with a minimum of work from the user, bringing a FORTh-like environment into the realm of practical scientific and business programming for the first time.

EDITOR'S RESPONSE TO
TRANSFORTH

The above material is extracted from explanatory sales material from the program vendor. Commentary we have indicated from TransFORTH users can be summarized:

1. This implementation should be named as one of the POLISHERS group of languages, as it compiles to assembly language rather than threaded code.
2. It is easier to add floating point math to FORTh, than to alter TransFORTH to use integers for execution speed improvements. Why not both?
3. If the implementor had done his DOS 3.3 Interface using the standard FORTh word BLOCK, an immense gain in value would result. Direct access and DOS compatibility.
4. <BUILDS DOES> probably could be added but apparently the implementor doesn't know how or chooses to deprive his customers of this powerful structure. Arrays are definitely not equivalent technically or philosophically.

In conclusion, it appears that TransFORTH is a reverse POLISH BASIC, with names rather than labels. A small amount of additional effort would have built upon FORTh, rather than strip out major attributes...
NEW PRODUCTS

FLEX-FORTH

Complete compiler/interpreter, assembler, editor, operating system for:
- APPLE II computers $25.00
- KIM computers $21.00

FLEX-FORTH is a complete structured language with compiler, interpreter, editor, assembler and operating system for any APPLE II or APPLE II+ computer with 48K and disk or KIM with 16K of memory. Most application programs run in less than 16K starting at 1000 HEX and often as little as 12K, including the FLEX-FORTH system itself.

This is a full-featured FORTH following the F.I.G. standard, and contains a 6502 assembler for encoding machine language algorithms if desired. The assembler permits macros BEGIN...UNTIL, BEGIN...AGAIN, BEGIN...WHILE..., REPEAT, IF...ENDIF, and IF...ELSE...ENDIF. Editor and virtual memory files are linked to the Apple DOS 3.2. An application note for upgrading to DOS 3.3 is included. Object code on disk with user manual sells for $25.00. (APPLE) or on cassette with user manual for $21.00 (KIM).

A complete source listing is available to purchasers of FLEX-FORTH for $20.00. The source is valuable in both showing how FORTH works and in giving examples of FORTH code and integrated assembly code.


MARX FORTH V1.1
Perkel Software Systems
1636 N. Sherman
Springfield, MO 65803
(417) 862-9830

Enhanced 8080 fig-FORTH implemented for Northstar System enhancements including link fields in front of name for fast compile speed; dynamic vocabulary relinking; case; arguments-results with 'to' variables; and more. 79-Standard package includes easy to use screen editor.

Price: $75.00

Smart assembler, meta-compiler and source code (in FORTH) sold separately. Call for information.

TWO NEW PRODUCTS FROM LAXEN AND HARRIS, INC.
Laxen and Harris, Inc.
26301 Southland Drive
Hayward, CA 94545
(415) 887-2894

1. Working FORTH
   Release 2.1

"Starting FORTH" compatible FORTH software for a 8080 or Z80 computer system with the CP/M (TM) operating system.

Copyright (C) 1981 by Laxen and Harris, Inc. All rights reserved.

This FORTH implementation is compatible with the popular book "Starting FORTH" by Leo Brodie. It is intended to be a companion to the book to aid learning FORTH. It is also a complete environment for developing and executing FORTH programs. It contains:
- Compilers
- Disk operating system
- Full names stored, up to 31 characters
- String handling
- Enhanced error checking
- 16-bit and 32-bit integer arithmetic and input/output

This is a single-user, single-task system which is not ROM-able as supplied. Floating point arithmetic and CP/M file access are not supported.

This system as supplied runs comfortably in a 8080 or Z80 computer system with at least 32K bytes of RAM memory, at least one floppy disk drive (8" single density, single sided, soft sectored format is assumed), and the "BIOS" part of the CP/M operating system. The use of a printer is supported but not required. This software may be easily modified to use other memory sizes or disk formats. It requires 14K bytes of memory which includes 5K bytes of disk buffers.

This FORTH system was adapted from the fig-FORTH model but is not fully compatible with that language dialect. It is also not fully compatible with the FORTH-79 Standard. The three dialects are similar, but the Starting-FORTH version has advantages over the other two.

Price: $35.00, plus $2.00 - Postage and Handling

CP/M is a registered trademark of Digital Research, Inc.

2. Learning FORTH

Learning FORTH is a computer aided instruction package that interactively teaches the student the fundamentals of the FORTH programming language and philosophy. It consists of a set of FORTH screens that contain program source code and instruction text. It is based on the book, "Starting FORTH" by Leo Brodie. It will run with any Starting FORTH compatible system, as well as fig-FORTH system. The manual is only one page long and describes how to load the system. After that, everything is self explanatory. It is supplied on 8" single density diskettes in IBM 3740 format. The price is $33.00 if ordered together with the Working FORTH Disk. Please add $2.00 for shipping and handling, and allow at least 3 weeks for delivery.

Note: Buy both for $55.00 plus $2.00 postage and handling.

POLYMORPHIC FORTH
Abstract Systems, etc.
1686 West Main Road
Portsmouth, RI 02871
(401) 683-0945
Ralph E. Kenyon, Jr.

Product Description: FORTH (Polymorphic fig-FORTH 1.1.0). 8080 fig-FORTH 1.1 without assembler or editor (uses Polymorphic resident editor.)

A demo application which computes a table of values for a general quadratic equation is included.

Polymorphic Systems 8813, 8815 only 16K. Documentation on FORTH not included.

Manual: documentation covers particular implementation details for fig-FORTH to interface to the Polymorphic Systems Microcomputer. Sorted VLIST included.

Implementation document available separately. Separate document available for cost of postage. Product data available on Polymorphic SSSD 5" diskette format. 4 copies sold to date. Price: $20.00, includes shipping, diskette, (R.I. residents add 6% sales tax). Warranty limited to replacement of a diskette damaged in shipment. (We'll try to fix any bugs discovered.) Orders shipped out within 3 days of receipt (usually next day).

HEATH H89 FORTH
MCA
8 Newfield Lane
Newtown, Conn. 06470

MCA announces the availability of FORTH for the Heath H89 computer. MCA FORTH is 8080 fig-FORTH V1.1 configured to run on a single disk H89 with 32K or more of memory, utilizing HDOS 1.6 or later.

MCA FORTH provides the standard FORTH facilities plus the following special features: HDOS file manipulation capability, a control character to restart FORTH (recover from boops), on-line tailoring of FORTH facilities (e.g., number of disk buffers), ability to hook to separately assembled routines, and use of Heath DRUG.

Items supplied with FORTH include the fig-Editor, an 8080 structured assembler, and two games provided as examples of FORTH programming.

The documentation supplied with MCA FORTH is suitable for experienced FORTH programmers; however, a bibliography of documentation for beginners is provided.

MCA FORTH is available from MCA on a 5-1/4" disk for $25 including documentation. Documentation is available for $4.00. (Conn. residents please add sales tax).
### NEW PRODUCTS FROM INNER ACCESS CORPORATION

1. **Fig-FORTH compiler/interpreter for PDP-11 or RT11**, RSX11M or stand-alone with source code in native assembler. Included in this package are an assembler and editor written in FORTH and installation documentation. **Price: $80.00**

   This is available on a one 8" single density diskette only.

2. **Reference Manual for PDP-11 fig-FORTH**. Includes an assembler and editor written in FORTH, density diskette only. **Price: $20.00**

3. **Finn-FORTH compiler/interpreter for CP/M or Cromemco CODS system** comes complete with source code written in native assembler. Included in this package are an assembler and editor written in FORTH and installation documentation. **Price: $80.00**

   All diskettes are single density, with 5-1/4" diskettes in 128 byte, 18 sector/track format and 8" diskettes in 128 byte, 26 sector/track (IBM) format.

   Released on two 5-1/4" diskettes with source in 8080 assembler.

   Released on one 8" diskette with source in 8080 assembler.

   Released on two 5-1/4" diskettes with source in Z80 assembler.

   Released on one 8" diskette with source in Z80 assembler.

   Manual for CP/M (or Cromemco) fig-FORTH available. **Price: $20.00**

4. **METAFOORTH™ Cross-compiler for CP/M or Cromemco CODS to produce 79-Standard FORTH on a target machine.** The target can include an application with dictionary heads and link words. It is available on single density diskettes with 128 byte 26 sector/track format. Target compilers may be readily produced for any of the following machines:

   - CROMEMCO-all models
   - TRS80 Model II under CP/M
   - Northstar Horizon
   - ProLog

   Released on two 5-1/4" diskettes or on one 8" diskette. **Price: $450.00**

5. **Complete Zilog (AVD) Z8002 Development Module fig-FORTH ROM set.** Contains 79-Standard FORTH with Z8002 assembler and editor in 4 (2716) PROMs. **Price: $250.00**

   CODE9
   Arthur M. Gorzki
   2240 S. Evanston Avenue
   Tulsa, OK 74114
   (918) 743-0113

   CODE9 is a M6809 Assembler for use with any fig-FORTH system. It features all M6809 addressing modes except long relative branch instructions. It performs syntax error checking at assembly time. Memory requirements: 4796 bytes free RAM above FORTH. CODE9 is distributed as a commented source listing and manual. **Price: $20.00**

   **PET-FORTH**
   by
   Datatronic AB
   Box 42994
   8-126 12 Stockholm
   Sweden
   (0) 8-744 59 20
   Peter Bengtson

   Product Description: Extended fig-FORTH for the Commodore CBM/PET computer series.

   Screen editor, utilizing the special CBM screen editing possibilities for compactness and ease of use, macro-assembler, double-precision extensions, CRT handling, random numbers, real-time clock, a very complete string package, IEEE control words, integer trig functions.

   An expansion disk (coming soon) will contain floating point arithmetic including complex numbers, transparent overlay control words for data and program segments, a file system, and more. A METAFOORTH compiler will shortly be available.

   Runs on CBM 8032 plus an 8050 dual disk drive. Other configurations coming: 4032, 4040, VIC, and MicroMainFrame.

   8032 version runs in 32K only. 4032 versions will run in either 16 or 32K.

   Manual Description: 322 pages, including all source code.

   Complete introduction to FORTH. Special chapters cover the assembler, (BUILD and DOES), IEEE handling, strings etc.

   Manual is available separately.

   Separate purchase price is $40.00. This is not creditable towards later purchase.

   Product/Ordering Data: Shipped as diskette and an accompanying security ROM, holding part of the Kernel.

   **Audio Tapes of 1980 FORTH Conference and 1980 FIG Convention**

   1. **FORTH-79 Discussion**, 200 min. **Price: $35.00**

   2. **Purpose of FIG**, 37 min. **Price: $10.00**

   3. **Charles Moore**, 63 min. **Price: $15.00**

   4. **FORTH, Alan Taylor**, 47 min. **Price: $15.00**

   Complete set $60.00

   edu-FORTH
   1442-A Walnut Street, #332
   Berkeley, CA 94709

   Currently, there are approximately 75 installations, after 2 months on the market.

   **Price: $290.00** includes diskette, ROM, manual, shipping and taxes.

   PET-FORTH, as all other Datatronic software, carries a life-time guarantee. All future versions will be distributed to the registered owners without any cost whatsoever.

   Delivery is immediate.

   US dealers are invited. UK sole distributor is Petaleat Electronic Services Ltd, 33/35 Portugal Road, Woking Surrey. You may also order directly from us.

   Diskette of FORTH Application Modules from
   Timin Engineering Company
   9575 Genesee Avenue, Ste. E-2
   San Diego, CA 92121
   (714) 455-9008

   The diskette of FORTH application modules, a new product by Timin Engineering, is a variety package of FORTH source code. It contains hundreds of FORTH definitions not previously published. Included on the diskette are data structures, software development aids, string manipulators, an expanded 32-bit vocabulary, a screen calculator, a typing practice program, and a menu generation/selection program. In addition, the diskette provides examples of recursion, BUILD...DOSE> usage, output number formatting, assembler definitions, and conversational programs. One hundred screens of software and one hundred screens of instructional documentation are supplied on the diskette. Every screen is in exemplary FORTH programming style.

   The FORTH screens, written by Scott Pickett, may be used with Timin FORTH or other fig-FORTH. The price for the diskette of FORTH application modules is $75.00 (if other than 8050 disk, add $15.00). To order the FORTH modules, write Timin Engineering Company, 9575 Genesee Avenue, Suite E-2, San Diego, CA 92121, or call (714) 455-9008.
The following vendors have versions of FORTH available or are consultants. (FIG makes no judgment on any products.)

**ALPHA MICRO**
Professional Management Services
724 Arasstradero Rd. #109
Palo Alto, CA 94306
(415) 858-2218

Sierra Computer Co.
617 Mark NE
Albuquerque, NM 87123

**APPLE**
IDPC Company
P. O. Box 11594
Philadelphia, PA 19116
(215) 676-3235

IUS (Cap’n Software)
281 Arlington Avenue
Berkeley, CA 94704
(415) 525-9452

George Lyons
280 Henderson St.
Jersey City, NJ 07302
(201) 451-2866

MicroMotion
12077 Wilshire Blvd. #506
Los Angeles, CA 90025
(213) 821-4340

**CROSS COMPILERS**
Nautilus Systems
P. O. Box 1098
Santa Cruz, CA 95061
(408) 475-7461

polyFORTH
FORTH, Inc.
2309 Pacific Coast Hwy.
Hermosa Beach, CA 90254
(213) 372-8493

LYNX
3301 Ocean Park #301
Santa Monica, CA 90405
(213) 450-2466

M & B Design
820 Sweetbay Drive
Sunnyvale, CA 94086

Micropolis
Shaw Labs, Ltd.
P. O. Box 3471
Hayward, CA 94540
(415) 276-6050

**North Star**
The Software Works, Inc.
P. O. Box 4386
Mountain View, CA 94040
(408) 736-4938

**POP-11**
Laboratory Software Systems, Inc.
3634 Mandeville Canyon Rd.
Los Angeles, CA 90069
(213) 472-6995

OSI
Consumer Computers
8907 La Mesa Blvd.
La Mesa, CA 92031
(714) 698-8088

Software Federation
44 University Dr.
Arlington Heights, IL 60004
(312) 259-1355

Technical Products Co.
P. O. Box 12983
Gainesville, FL 32604
(904) 372-8439

Tom Zimmer
292 Falletto Dr.
Mipitas, CA 90305

1802
FSS
P. O. Box 8403
Austin, TX 78712
(512) 477-2207

6800 & 6809
Kenyon Microsystems
1927 Curtis Avenue
Redondo Beach, CA 90278
(213) 376-9941

TRS-80
The Micro Works
P. O. Box 1110
Del Mar, CA 92014
(714) 947-2400

Miller Microcomputer Services
61 Lake Shore Rd.
Natolok, MA 01760
(617) 635-6136

The Software Farm
P. O. Box 2304
Reston, VA 22090

Sirius Systems
7528 Oak Ridge Hwy.
Knoxville, TN 37921
(615) 693-6583

6502
Eric C. Rehak
540 S. Ranch View Circle #61
Anaheim Hills, CA 92807

Saturn Software, Ltd.
P. O. Box 397
New Westminster, BC
V3L 4Y7 CANADA

8800/7800/CPU/M
Laboratory Microsystems
4147 Beethoven St.
Los Angeles, CA 90066
(213) 390-9292

Timin Engineering Co.
9575 Geneve Ave. #E-2
San Diego, CA 92121
(714) 455-9000

**Application Packages**
Innovsys
2150 Shattuck Avenue
Berkeley, CA 94704
(415) 843-8114

Decision Resources Corp.
28203 Ridgefern Ct.
Rancho Palermo, CA 90274
(213) 377-3533

68000
Emperical Res. Grp.
P. O. Box 1176
Milton, WA 98344
(206) 631-4855

**Firmware, Boards and Machines**
Datatron
7911 NE 33rd Dr.
Portland, OR 97211
(503) 284-8277

Forward Technology
2595 Martin Avenue
Santa Clara, CA 95050
(408) 293-8993

Rockwell International Microelectronics Devices
P. O. Box 5669
Anaheim, CA 92803
(714) 632-2862

Zandex Corp.
5938 Daugherty Rd.
Dublin, CA 94566

**Variety of FORTH Products**
Interactive Computer Systems, Inc.
6403 Di Marco Rd.
Tampa, FL 33614

Mountain View Press
P. O. Box 4656
Mountain View, CA 94040
(415) 961-4103

Supersoft Associates
P. O. Box 1628
Champaign, IL 61820
(217) 359-2112

**Consultants**
Creative Solutions, Inc.
4001 Randolph Rd.
Rockville, MD 20852

Dave Boulton
581 Oakridge Dr.
Redwood City, CA 94062
(415) 368-5257

Go FORTH
504 Lakemead Way
Redwood City, CA 94062
(415) 566-6124

Inner Access
517K Marine View
Belmont, CA 94002
(651) 591-8299

John S. James
P. O. Box 348
Berkeley, CA 94701

Laxen & Harris, Inc.
24301 Southland Drive, #303
Hayward, CA 94545
(415) 887-2894

Microsystems, Inc.
2500 E. Foothill Blvd., #102
Pasadena, CA 91107
(213) 577-1471
<table>
<thead>
<tr>
<th>Item</th>
<th>USA</th>
<th>FOREIGN AIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Membership in FORTH INTEREST GROUP and Volume III (6 issues) of</td>
<td>$15</td>
<td>$27</td>
</tr>
<tr>
<td>FORTH DIMENSIONS.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Volume II of FORTH DIMENSIONS (6 issues)</td>
<td>$15</td>
<td>$18</td>
</tr>
<tr>
<td>Volume I of FORTH DIMENSIONS (6 issues)</td>
<td>$15</td>
<td>$18</td>
</tr>
<tr>
<td>fig-FORTH Installation Manual, containing the language model of</td>
<td>$15</td>
<td>$18</td>
</tr>
<tr>
<td>fig-FORTH, a complete glossary, memory map and installation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>instructions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembly Language Source Listings of fig-FORTH for specific CPU's</td>
<td></td>
<td></td>
</tr>
<tr>
<td>and machines.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The above manual is required for installation. Check appropriate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>box(es).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price per each.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1802</td>
<td>$15</td>
<td>$18</td>
</tr>
<tr>
<td>6502</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6809</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8080</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8086/8088</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APPLE II</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PACE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALPHA MICRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDP-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOVA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>“Starting FORTH” by Brodie. BEST book on FORTH. (Paperback) NEW</td>
<td>$16</td>
<td>$20</td>
</tr>
<tr>
<td>“Starting FORTH” by Brodie. (Hard Cover) E</td>
<td>$20</td>
<td>$25</td>
</tr>
<tr>
<td>PROCEEDINGS 1980 FORML (FORTH Modification Lab) Conference</td>
<td>$25</td>
<td>$35</td>
</tr>
<tr>
<td>PROCEEDINGS 1981 FORTH Univ. of Rochester Conference</td>
<td>$25</td>
<td>$35</td>
</tr>
<tr>
<td>PROCEEDINGS 1981 FORML Conference, Both Volumes NEW</td>
<td>$40</td>
<td>$55</td>
</tr>
<tr>
<td>Volume I, Language Structure</td>
<td>$25</td>
<td>$35</td>
</tr>
<tr>
<td>Volume II, Systems and Applications</td>
<td>$25</td>
<td>$35</td>
</tr>
<tr>
<td>FORTH-79 Standard, a publication of the FORTH Standards Team</td>
<td>$15</td>
<td>$18</td>
</tr>
<tr>
<td>Kitt Peak Primer, by Stevens. An indepth self-study primer.</td>
<td>$25</td>
<td>$35</td>
</tr>
<tr>
<td>BYTE Magazine Reprints of FORTH articles, 8/80 to 4/81</td>
<td>$ 5</td>
<td>$10</td>
</tr>
<tr>
<td>FIG T-shirts: Small Medium Large X-Large</td>
<td>$10</td>
<td>$12</td>
</tr>
<tr>
<td>Poster, Aug 1980 BYTE cover, 16 x 22”</td>
<td>$ 3</td>
<td>$ 5</td>
</tr>
<tr>
<td>FORTH Programmer’s Reference Card. If ordered separately, send a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stamped, addressed envelope. FREE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Make check or money order in US Funds on US bank, payable to: FIG. All prices include postage. *No purchase orders without check.*

ORDER PHONE NUMBER: (415) 962-8653

FORTH INTEREST GROUP PO BOX 1105 SAN CARLOS, CA 94070
Support Your Local FIG Chapter