


## Published by Forth Interest Group

Volume III No. 1
May/June 1981
Publisher
Guest Editor
Roy C. Martens
C. J. Street

Editorial Review Board
Bill Ragsdale
Dave Boulton
Kim Harris
John James
Dave Kilbridge
Henry Laxen
George Maverick
Bob Smith
John Bumgarner
FORTH DIMENSIONS solicits editorial material, comments and ietters. No responsibility is assumed for accuracy of material submitted. ALL MATERIAL PUBLISHED BY THE FORTH INTEREST GROUP IS IN THE PUBLIC DOMAIN. Information in FORTH DIMENSIONS may be reproduced with credit given to the author and the Forth Interest Group.

Subscription to FORTH DIMENSIONS is free with membership in the Forth Interest Group at $\$ 12.00$ per year (\$24.00 foreign air). For membership, change of address and/or to submit material, the address is:

## Forth Interest Group

P.O. Box 1105

San Carlos. CA 94070

## 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 witn a vailable 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.

## EDITOR'S COLUMN

The last edition of FORTH DIMENSIONS was the beginning of many changes in editorial policy and format. All these changes are designed to make FORTH DIMENSIONS a practical and useful communications tool.

This practical approach continues. In this edition you will find a number of utility programs that will make the task of implementing practical applications in FORTH easier and faster. All of these utilities have been contributed by FIG members who have found them to be valuable tools. The editorial staff thanks these contributors and would like to encourage all FIG members to share their ideas and experience.

If you have a programming idea or tool that you have found useful, please send it to

Editor
FORTH DIMENSIONS
P.O. Box 1105

San Carlos. CA 94070
YOU DON'T HAVE TO BE A WAITER-the editorial staff will provide whatever copywriting is necessary to make your ideas publishable.

On the aesthetic side, you will find this edition contains several photographs and art illustrations. This is a FORTH DIMENSIONS' first and you can expect to see more in the future. Photographs and art illustrations will be chosen and published on the basis of their educational and human interest value.

This issue also introduces the character HEX who will be FORTH DIMENSIONS' official comic strip. You will find the HEX comic strip in future editions of FORTH DIMENSIONS. HEX's adventures will be a combination of humor and education. Ideas for HEX comic strips are welcome.

## C. J. Street

Editor

## PUBLISHER'S COLUMN

Renewals and new members are coming in at a fast pace. We expect to climb to 3,000 members in the next few months and to 5,000 within a year.

Both the Computer Faire in San Francisco and the Computer Festival in Trenton, New Jersey were huge successes. We'll be in both again next year. (l'd like to know of any other shows where you think FIG should exhibit.)

Good material is coming in for FORTH DIMENSIONS. Keep it coming and send in your comments.

## Roy Martens

## LETTERS

Dear Fig:
My company is developing business systems using FORTH and we would be interested in communicating with local FIGGIES as well as offering our input to FST, FORM, FIC and other applicable "r" acronyms since it is obviously in our interest to promote the spread and acceptance of FORTH. We're also confirmed FORTH fanatics.

```
David B. Moens
BUSINESS SYSTEMS SOFTWARE, IMC.
27 East Rings Highway
Haddonfield, NJ 08033
(609) 429-0229
```

You are our kind of fanatic and we're happy to put the word out for you Dave! -ed.

Dear Fig:
Re: "Born-again programmer" and "Bornagain FIGGER" in FORTH DIMENSIONS II/5.

My interest in FORTH as a programming language does not envolve becoming mired down in the morass of a religion. It would be best to stay on rational grounds in the development of FORTH and leave religion to those who are unable to think without faith.

I will not take part in a religious group. Therefore I will not be renewing my membership.

Larry R. Shultis<br>P.O. Box 218<br>Fontana, WI 53125

Just goes to show you that there is more than one type of fanatic! Reep the faith, Larry, (OOPS, sorry about that! I meant: Don't worry, Larry,) FIG is not envolving (your spelling) into a religious group. -ed.

Dear Fig:
Thank you for the prompt and efficient service $I$ have received. I realize that you can't have mach time left to look after the rest of the world, but without your interest it may never have reached these shores. Spread the good WORD.
J. Huteley UNIVERSITY OF AUCELAND
19 Duncan Avenue
Auckland 8 , New Zealand

You are very welcome! -a ed.

Editor' note:
At the WEST COAST COMPUTER FAIRE in California two versions of a PORTH bumper strip were circulated:
?FORTH IF HONK THEN
or alternately
: LOVE-FORTH
IF HONR THEN ;
Just thought you might like to know. -ed.

Dear Fig:
TGIF is very interested in swapping listings with other Fig-FORTH groups. Our current listings are 2 Decompilers; Full screen editor; CASE statements for 8080, Z80 and 6502; 6502 Assembler.

```
TGIF
FORTH INTEREST GROUP -- TULSA
Box 1133
Tulsa, OR 74103
```

How about sending them in to FORTH DIMENSIONS? -- ed.

## Dear Fig:

I was lucky enough to attend one day of the recent West Coast Computer Faire and to meet some of the mentors of FIG. I had numerous questions and enjoyed talking to Bill Ragsdale and others about them. (By the way, for those of you who couldn't make it to the Faire, the FIG booth was one of the most crowded. People were standing there like no where else even as the 5:00 closing approached. We all owe a thanks to the dedicated folks for their time and effort in this endeavor, promoting and spreading the word of FORTH and FIG.)

One of my questions to Bill was "How can we remote members contribute to FIG" in ways other than articles for FORTH DIMENSIONS. I got a number of project ideas, for one of which I need the help of the whole membership. So PLEASE: NOW HERE THIS!

I propose to assemble a book of utility program packages for publication by FIG. I have a list of functions which I think should be included. This covers such things as editors (both the current FIG line oriented editor and a video screen type editor), string processing, data structures, extended math (double precision as an example), math functions (sin, $\log$, etc.), matrix operations, and floating point routines. No doubt there are others to be considered and I solicit your suggestions.

The plan $I$ propose to compile and publish such a document is as follows:

1. Members are asked to send their proposals for implementing utility packages to me at the above address (or through FIG). These proposals should consist of well documented (lots of comments) fig-FORTH source code accompanied by complete instructions for use, any known operating limitations, and a brief technical description or reference if appropriate. The programs should be as transportable as practicable; if system unique routines are necessary,
enough information should be provided so as to allow them to be adapted to a different machine.
2. I will compile a draft of the proposed publication and submit it to a technical review committee for review and appropriate testing. This committee of FIG members (I am looking for volunteers) will scrutinize the proposals (and alternatives if appropriate), test them on a running FORTH system, and make recomendations.
3. I will then compile the final version and submit it to FIG for publication.

I have set some timelines for compiling this compendium such that it can be published by next year's Computer Faire. Material should be sent in no later then 1 September 1981 (please send them early and give me a chance to get ahead). I will distribute the draft for review and testing by 15 October 1981. Finally I will begin compiling the final version by 1 January 1982 and have it ready for publication by 1 February 1982.

This may seem like a long time abuilding, but $I$ want to provide ample opportunity for the contributors to develop their ideas fully and conduct a fair amount of testing themselves before submission, I also want to provide a good review by the committee to provide the highest quality document for FIG. It will be worth it in the long run. Your contributions will be sincerely appreciated, and though the publication, as are all of FIG's, will be in the public domain, credit will be given to the authors and contributors. So if nothing else, this is a chance to get your name in print, in an international publication.

Sincerely,
George 0. Young III
617 Mark NE
Albuquerque, NM 87123

This is a great project. Our goals continue to be to decentralize FIG activities, and George's project of a published "Goodies Package" will be helpful to all. Contributors should send a brief description to George and then prepare the full document. This will allow co-ordination of similar material. -- ed.


Spreading the fig-FORTH at the West Coast Computer Faire, April, 1981.
Top: (1-r) Michel Mannoni (FORTH vendor), Dave Boulton and Martin Schaaf (Answering the question: What's FORTH? 10,000 times) Bottom: (1-r) Roy Martens (FD Publisher), Anne \& Bill Ragsdale (FIG prime movers). Order your T-shirt, like Bill's!

## ANNOUNCEMENTS

FORML CONFERENCE CALL FOR PAPERS

Papers are requested for the three day technical workshop to be held next November 25th to 27 th at the Asilomar Conference Grounds in Pacific Grove, California (Monterey Peninsula).

Although registration for this conference is not yet open, attendance will be limited to 60 persons. Authors will be accepted before listeners, so if you want to attend, the only sure way is to write a paper. Please note: abstracts or proposals for papers or discussions must be submitted no later than September 1, 1981 for inclusion in the conference and its proceedings; completed papers by September 15, 1981.

The purpose of this workshop is to discuss advanced technical topics related to FORTH implementation, language and its applications. Potential authors should write for an author's packet for detailed instructions. Send all correspondence regarding the conference or papers to:

FORML CONFERENCE
P.O. Box 51351

Palo Alto, CA 94303

## FORTH WORKSHOPS

Beginners and advanced workshops in FORTH are being jointly sponsored by the College of Notre Dame and Inner Access Corporation both of Belmont, CA.

Beginners workshops start June 23 and advanced workshops start July 14. Classes meet every Tuesday and Thursday evening from 7:00 to 9:00 P.M. Registration is $\$ 135$ for 3 weeks ( 12 hours).

For more information and registration contact:

College of Notre Dame (415) 593-1601

## CONTINUING DIALOG ON

## FORTH－79 STANDARD

Dear Bill：
We recently obtained a copy of the FORTH－70 Standard from FIG and are attempting to align our version of FORTH with it．The document is generally well done and in most cases clearly and con－ cisely expressed．However，there are about a half dozen or so definitions that seem to us somewhat ambiguous．

I am writing to you in the hopes that you can clarify the word definitions in question；or，that you can refer us to someone who can．I am also interested in knowing whether the FIG model has been aligned yet，if not，when it will be．

My list of questions is enclosed and I would appreciate anything you can do to assist us in their resolution．

Cordially，
Robert D．Villwock
MICROSYSTEMS，INC．
2500 E．Foothill Blvd．， 102
Pasadena，CA 91107
OPEN QUESTIONS
FORTH－79 Standard
1．For the words $/$ and $* /$ does the terminology＂rounded toward zero＂ essentially mean truncated？If not， precisely what does it mean？

2．The word SGIN is now apparently defined to be used＂outside＂of the $\left\langle\begin{array}{l}\text { \＃}\end{array}\right.$ 〉operators．What is the pre－ cise definition of where the minus sign character is to be stored？Why was this word changed from its former function between 〈\＃and $\#$ 〉？

3．The word＇：＇is defined as a non－ precedence word．Is this a typo or is it intentional？If intentional， could you explain the rationale？It seems that the number of occasions for which＂colon＂needs to be com－
piled are few and could easily be handled by using the［COMPILE］ operator．ON the other hand，syntax errors and typos often result in mistaken attempts to compile＇：＇ which，when it＇s an immediate word， can be flagged by the compiler．

4．The word CODE is defined as using the form：

CODE 〈name〉 ．．．END－CODE
However，the word ；CODE says nothing about the corresponding form．Our version of FORTH rquires that code level action routines defined by ；CODE also be terminated by the word END－CODE．Is this compatible with FORTH－79？

5．The words FIND，＇，＇＇：＇，etc．，as defined in the Standard，indicated a search of CONTEXT and FORTH only．Is it considered an incompatibility if the CURRENT vocabulary is also searched（if different）？The defini－ tion of VOCABULARY is not clear regarding the possibility of＂sub－ vocabularies＂such as ABC chained to XYZ chained to FORTH．If this is allowed，and，$A B C$ is the CONTEXT vocabulary，is not $A B C, X Y Z$ ，and FORTH searched？

6．What is the memonic significance of the $C$ words such as C！，CMOVE， etc．？Surely it doesn＇t stand for ＂cell，＂does it？The term＂cell＂is defined on page 3 of the Standard to be a 16－bit memory location．The word MOVE is defined on page 26 to trans－ fer 16－bit words（＂cells＂），while the word CHOVE on page 20 is defined to move bytes（not＂cells＂）．If the C does stand for＂cell＂what is the rationale？Why was the former stan－ dard＇s $B$（for byte）replaced by the mysterious C？

7．I note that in the reference section of the Standard，the word DPL which formerly used to handle both input and output＂point＂situations now strongly emphasizes that input con－ versations should not affect its
value．What is the reason for this restriction？How likely is it that this may become part of the Standard？

8．The definition for CREATE is not clear．Does the second sentence ＂When 〈name〉 is subsequently executed，the address of the first byte of＜name〉＇s parameter field is left on the stack＂mean that the word CREATE alone is to function this way or only when followed by ；CODE or DOES $>$ ？In other words，is it intended that CREATE work as in the FIG model or has its definition changed？Taken literally，FORTH－79 says that CREATE will generate an unsmudged header with the CFA point－ ing to the run time procedure for variables．Is this what is intended？

COMMENTARY FROM THE FORTH DOCTOR

1．Some computers apparently（by Stan－ dard Team comment）round quotients and remainders to smaller magnitude （more negative）．Trucation of nega－ tive quotients would do this．If a correct representation is not possi－ ble，the result should be nearer zero．Dave Boulton is more know－ ledgeable on this point．

2．＂Sign is to be used within 〈＊and \＃$>$ ．The user chooses where to store the sign．Notice that no word gener－ ates the saving of the sign．In fig－ FORTH the only difference is the ROT would be explicitly done just before SIGN，rather than in SIGN．

3．FIG and the Europeans make ：an imme－ diate word for error control．Other users，and FORTH，Inc．reject this level of error control－－too bad！We need a technical paper presenting the trade－offs（code needed and compila－ tion slowdown）．Conversation at a team meeting is insufficient to change opinions developed over ten years．

4．These topics were barely touched on by the Team as CODE definitions are not portable．；CODE probably should
terminate in END－CODE．This is an unresolved area．

5．The standard wording was painstaking－ ly done regarding vocabularies．This is the most divergent topic among users．all known methods can comply with the Standard，but it does less than all systems．The rationale is that you build CURRENT but you exe－ cute only from CONTEXT（and FORTH）． No chaining is recognized，beyond context leading to FORTA．This may be physical links or logical（within FIND）．Again，position papers are essential to get a common，more advanced，construct．

6．Charles Moore has used $C$ for ten years as a character（byte）prefix． Ignore（if you can）that a character is defined as 7 bits in the stan－ dard．This was a hotly disputed point with FIG and the Europeans for ＂B＂yte and FORTH，Inc．and a couple of others for＂C＂．Kitt Peak was adamant before the meeting for＂$B$＂ and other uniformity improvements． Their representatives made no defense of the issue．Historical precedence wins this one．

7．Reference Section is just left－ overs．Only one vote of any team member was sufficient to maintain a Reference word on the list．The Standard attempts to minimize system variables．Increased usage of special variables is unlikely． Things like DPS are delegated to applications．

8．The definition of CREATE is quite clear．You have stated it and then correctly paraphrased it．Other defining words may be used before DOES $>$ which help build a parameter field．DOES＞rewrites the code field to its own code．

$$
\text { : CPU CONSTANT DOES }>\text {; }
$$

is equivalent to

$$
\text { : CPU CREATE , DOES }>
$$

## TECHNICAL NOTES, BUGS \& FIXES

## Dear Fig:

I have recently brought up FORTH on a 6800 system and find it to be a very easy and powerful system for microcomputers.

I have a mini-computer with a crossassembler on it which I used to assemble the source after keying it in. Naturally, as soon as I got it working I wanted to change it. I feel that the EXPECT routine and backspace handling could be improved significantly by incorporation of the enclosed recommendations.

I also experimented with the GLOSSARY routine submitted by D.W. Borden in FORTH DIMENSIONS, Volume 1, No. 4. I modified it to handle the variable length name field and changed the format slightly.

Keep up the good work.
Toby L. Rraft
San Diego

Modifications to the fig-FORTH boot-up 1iterals:

1. Backspace Character

Character to emit in response to a backspace entry. X'08' ( control-H ) is character FORTH responds to for backspace function. Character to emit is terminal dependent and should be defined in the user table.

This also allows use of a printable character (e.g. C'\' ) to emit for backspace for use on printing terminals.
2. Form Feed Character

Character to emit to cause terminal ( or printer ) to advance to top of form. This is also device dependent and should be in user table.
3. Form Feed Delay

Number of null characters to emit after issuing a form feed character. This is similar to CR/LF delay which is already provided.

## Recommendation :

Add variable 'BSTOF' to user table.
$X^{\prime} B B F F^{\prime}$ - two characters of data
FF - form feed character ( $X^{\prime} O^{\prime}$ initial value)
BB - back space character ( $X^{\prime} 08^{\prime}$ initial value )

Add word 'BSTOF' to vocabulary to access this variable in user table. ( Similar to 'BASE' )

Modify definition of current user variable 'DELAY' to include formfeed delay in upper byte.

Add word 'DELAY' to vocabulary to access this variable in user table.

Modify startup parameters and cold start accordingly.

Modify EXPECT to use user defined backspace character and to explicitly generate bell code ( $X^{\prime} 07^{\prime}$ ). Currently, EXPECT tests for the beginning of the buffer and subtracts the boolean flag result from X'08' to generate the character to emit in response to a backspace.

Toby L. Kraft
7822 Convoy Court
San Diego, CA 92111
(714) 268-3390

This really needs expansion and generality. How about terminals that need an "escape sequence" to clear screen, i.e. form feed? Toby, HEX should be used insteat of $X^{\prime}$. --ed.

Dear Fig:
I wish to convey a concept which has greatly increased the clarity of my FORTH coding. It has to do with in-line documentation of the contents of the stack (comments within parathesis).

Unfortunately, none of the existing techniques (space, hyphens, brackets, or ordinal suffix) provide the brevity and clarity that one becomes accustomed to with FORTH. The technique which I have devised provides both. It revolves around the backslash character ' $\$ ', which I refer to as 'under' and the double hyphen '- ', which I refer to as 'leaves'. Using this terminology, the following comment:
( address \count -- )
is read "address under count leaves nothing," and

$$
(N 1 \backslash N 2-N 3)
$$

is read "Numberl under Number2 leaves Number 3."

The 'under' symbol imparts a clear verbal and graphic representation of the ordering of the stack contents, and provides an elegant solution to a major problem encountered when transporting FORTH algorithms and source code.

Don Colburn
Creative Solutions, Inc. 4801 Randolph Road Rockville, MD 20852

## Dear Fig:

Some time ago I bought your Installation Manual and the 6502 Assembly Listing. I have been studying both for quite a while, and am also a charter member of the Potomac FORTH Interest Group (PFIG: Joel Shprentz and Paul VanDerEijk).

I have MMS FORTH (cassette) for the TRS80 up, and have just bought GEOTEC FLEX-FORTH for my RIM, although I don't have my 16X ram card installed in KIM yet. I do like FORTH!!! The PFIG has been fairly inactive for some time due to lack of a meeting place, but Joel Shprentz has been conducting some Intermediate FORTH classes ( $\$ 30$ for six lessons) which are ongoing, and very interesting - we are well into 〈BUILDS/DOES〉, and will then go on to disking, etc. Ask Joel for details.

I'm still planning to bring up FORTH on the KIM from my own hand-assembled version, just to satisfy my own curiousity about what makes FORTH tick. I do think I'm finally beginning to understand how everything fits together.

In this vein, I have a few comments to pass on from an (advancing) novice FORTH enthusiast. The first two comments regard the above referenced Installation Manual and 6502 Assembly Listing. The last two are ideas of my own which I offer for what they are worth.

1. There is a disparity in the Installation Manual version of the 6502 memory map regarding the placement of the Disk buffer and Dser Area.

Indeed, there is disparity in the 6502 Assembly Listing between what is done near the front and what is actually implemented (per the installation Manual). The Installation Manual puts the Disk buffer at the top of RAM with the User area just below. Line 0051 of the assembly manual says User area is top 128 bytes, with disk buffer next (line 0052). CREATE assumes just the oppposite in both the Installation Manual and Assembly Listing. (Editor -- correct on all all points. The author was inconsistent.)
2. In screen 49 of the Installation Manual, I see no need whatsoever for a dedicated word such as ID. to move the word name to Pad and then type it out! The first 4 words are not needed, and neither are the words following " - " (PAD SWAP CMOVE PAD). Just a waste of time and space to bring the name to PAD and then type it out! (Editor -- this is not so. If you have WIDTH set to less than 31 , ID. is required.)
3. I would suggest a word ( $Q$ ) that might be inserted into any type of loop (DO/LOOP or BEGIN/AGAIN) to allow a timely exit when things go awry (as they do with Novices!). It's very simple - : $Q$ ?TERMINAL IF QUIT ENDIF ; MMS FORTH has this embedded into the code of " : ", but I think that's overkill. But it sure is nice to undo errors put into loops. (Editor -- this is terrible style. LEAVE is the correct way for a controlled termination.)
4. This has specifically to do with the Jump Indirect of the 6502 as used in both the Installation Manual and the assembly listing. Having used the 6502 for better than 4 years, I have yet to use the JMP indirect after finding out about its shortcoming of wrapping around within a page if low byte of address is SFF. I pretend this opcode does not exist. (Editor -- CREATE on 6502 systems correctly
places code field. Anymore comments should be directed to Chuck Peddle, designer of 6502.)

Keep up the good work.
Edward B. (Ted) Beach 5112 Williamsburg Blvd. Arlington, VA 22207

## CORRECTION ON SEARCH

## by John James

(Vo1. II *6)
When you are debugging or modifying a program, it is often important to search the whole program text, or a range of it, for a given string (e.g. an operation name). The 'SEARCH' operation given below does this.

To use 'SEARCH', you need to have the FIG editor running already. This is because 'SEARCH' uses some of the editor operations in its own definition. The 'SEARCH' source code fits easily into a single screen; it is so short because it uses the already-defined editing functions. Incidently, the FIG editor is documented and listed in the back of FIG's Installation Manual.

Use the editor to store the source code of 'SEARCH' onto a screen. Then when you need to search, load the screen. (Of course if you are using a proprietary version of FORTH, it may have an editor and search function built in and automatically available when needed. This article-ette is mainly for FORTH users whose systems are the ten-dollar type-it-in-yourself variety).

Here is an example of using 'SEARCH'. We are searching for the string 'COUNT' in screens 39-41; the source code of 'SEARCH' is on screen 40. The screen and line numbers are shown for each hit. Incidently, the search string may contain blanks. Just type the first screen number, the last screen number, 'SEARCH' followed by one blank and the target text string. Conclude the line with return. The routine will scan over the range of
screens doing a text match for the target string. All matches will be listed with the line number and screen number.

## Happy SEARCHing!

3941 SEARCH OUNT
00 VARIABIE COUNT ER 240
1 OUNT ER +! COUNIER © 440
1 CONIER + ! COUNI ER © 440
$56>$ IF 0 CONTI ER! 540 12 EMIT 01 TEXI ${ }^{-} 0$ COUNT ER! 840 OR

## SCR 10

- \{ search, over ramge of screems (ufk)
- decinal

2 oo variaille counter
: JuMp ( THE LIME NUMPER AMD MANDLE PAGIMS)
1 COUHTER 1 COUMTER
56 ) IF O COUNTER:
CR CR IS MESSAGE 12 EMII THEN :
: SEARCH ( FROM, to -.- Target StRING)
12 EnIT OI TEXT O COUMTER
is SUAP DO FORTM I SCR :
EDITOR TOP
begin llime if on scr p oum tien
1023 RI ( UNYIL
loop ; Cr ." search is loadet - ;
IS TYPICAL USE TO LDCATE 'KEY-HORD': 21 44 SEARCH KEY-dome

## PROGRAMMING AIDS \& UTILITIES

Rim Harris<br>FORTHRIGHT ENTERPRISES<br>P.O. Box 50911<br>Palo Alto, CA 94303

In true ideal FORTH programing style the definitions contained within the screens clearly designates their use.

81 LIST 82 LIST
SCR * 81
0 (Software Development tools fig-FORTH 1.x)
1 (for LOADing)
2 : LOAD ( scri - ) DUP . LOAD ;
3 : THEU ( lstScreent lastScreen - )
( LOADs range of screens )
4 I+ SWAP DO I LOAD LOOP ;
5
6 ( non-destructive stack print )
7 : DEPTH ( - \#StackCellsUsed) SPe SO © SNAP - $2 /$;

8 : .S (prints stack contents, top last; stack unchanged)
9 DNPH TF SPe 2- SOE2- DO I? $-2+1008$
10 ELSE " Enpty" THAN ;
11
12 (which vocabulary is being referenced?)
13 : .vOC (prints CONDEXT VOCABULARI name)
14 Coviexre e 4 - NRA D. ;
15

```
SCR * 82
    O (Tools: numer printing fig-0xH1.x)
    1
    2: .BASE (-) (prints arrrent radix in
        decimal)
    3 BASE O DUP DECNML . BASE! ;
    4
    5 ( create bese-specific stack-print operators)
    6 : BASED. (\BUILDS: nembese -)
        ( DOES: n- )
            BUIDS>
    8 DOE >e BASE E SNMP BISE! SWAP.
        BASE ! ;
    9
10 16 BASED. H. (print top-of-stack in hex )
11 8 BaSED. O. (print in octal)
12 2 BISED. B. (print in binery)
1 3
14
15
```

ar

The following utility indexes 10 screens at a time and is an excellent aid in searching.

HEX : + INDEX 1130 DO
DUP 10+ SWAP OVER INDEX
KEY ?ESC IF LEAVE THEN LOOP;

The following utility was contributed by Sam Bassett and is an excellent program development aid that shows you what the current base is

```
: BASE?
```

BASE
DUP
DECIMAL
-
BASE !

Here is an adaptation of George Shaw's VIEW to use the word WHERE, which on my system invokes a full screeen editor that highlights the word pointed to by a block number and displacement. It certainly helps pick out a word in dense code.

SCR \# 66


## HELP WANTED

## Senior Level FORTH Programers

Friends-Amis
505 Beach Street
San Francisco, CA 94133
Call: Tom Buckholtz
(415) 928-2800

Intermediate \& Senior Level FORTH Programmers for Data Entry Applications

MSI Data
340 Fischer Avenue
Costa Mesa, CA 92627
Call: Joan Ramstedt
(714) 549-6125

## PRODUCT REVIEW

by C.H. Ting, Feb. 26, 1981
Timin-FORTH, from Mitchel E. Timin Engineering Co., 9575 Genesee Ave., Suite E2, San Diego; CA 92121, (714) 455-9008. $8^{\prime \prime}$ single density diskette, $\$ 95.00$

I was invited by Dr. Timin to compare his CP/M FORTH (FD II/3, P. 56) with the 2-80 FORTH by Ray Duncan, Laboratory Microsystems (FD II/3, P. 54; FD II/5, p. 145) I ran the two FORTH systems on his home made $2-80$ computer ( $\mathrm{S}-100$ bus, 6 MHz ) The results of a few bench marks were:

| Program | Timin | 2-80 |
| :---: | :---: | :---: |
| : LOOPIESI TFFF 0 DO LOOP; | 2.3 sec | 9 sec |
| : -TEST TFFF 0 D I DUP - DEOP LOOP; | 5.9 | 7.4 |
| : *TEST 7FFP 0 DO I DOP * DRDP LOOP | 44.0 | 54.9 |
| :/TEST 7FFP 0 D 7 TFF I / DEOP LOP | ; 74.3 | 88.6 |
| : WIPE 12061 D I CIEAR LOOP ; | 34.3 | 81.8 |
| 97 IRAD ( four huntred eighty 9's ) | 17.9 | 18.6 |

$\zeta$
I was surprised that Timin-FORTH which is 8080 fig-FORTH ran faster than $Z-80$ FORTH which uses the extra $2-80$ registers for IP and W. Dr. Timin's opinion was that the $\mathrm{z}-80$ instructions using these extra registers are slower then the simpler 8080 instructions. The word WIPE tests disc access time. Timin-FORTH accesses the disc by 1024 byte blocks, and it is twice as fast as $\mathrm{Z}-80$ FORTH, which reads/writes by 128 byte sectors, as in the fig-FORTH model.

The dictionary in Timin-FORTH is about 11 Kbytes, including an editor and an assembler. The editor is the same as that of the fig-FORTH model. The assembler has all the 2-80 instructions. An interesting word SAVE allows the whole system including application words to be preserved
as a CP/M file which can be loaded back for execution. It maintains eight 1 Kbyte disc buffers.

The documentation supplied with the system is a 68 page booklet 'USER'S MANUAL $\&$ tuTORIAL'. It is a very well done manual introducing users to the systems and to the FORTH language. However, source listings are not provided.

My overall impresssion was that this is a well rounded FORTH system suitable for engineering and professional applications.

Editors Comment -- FORTH Dimensions refrains from publishing timing benchmarks as this reflects processor speed more than effectiveness of problem solving. However, the above review points out that the allegedly superior $2-80$ runs these tests slower than the 8080. Our point is that the user should evaluate all aspects of problem solving: hardware characteristics, language implementation and application technique. The Timin manual is sold separatly for $\$ 20.00$. This price is not justified by the copy received for our evaluation.

## HELP WANTED

## FORTH PROGRAMMER

PDP-11 RSX Op Sys on Site Contractor
Micro/Temps
790 Lucerne Dr.
Sunnyvale, CA 94086
(408) 738-4100

FORTH TELE-CONFERENCE IS NOW OPERATIONAL
FORTH now has a dynamic, public access data base. By dialing into the FIG Communitree (tw, the CommuniTree Group) you may access our tele-conferencing system. It was created by Figger John James to allow group interaction to build upon our collective knowledge.

The number is 415-538-3580. The system runs 24 hours a day. Use a 300 baud modem and start with two "returns", the system is self-instructing. This conference holds information on employment, vendors, applications, announcement calendar, inquiries, books, etc. Information of the conference is organized in a tree structure, hence the name " Conference Tree".

Our hope is that half of the callers will review the available material and then ask questions. The other half should add answers to these questions. You simply find a topic or message and attach your query/response. Users naturally organize their material in a form that facilitates retrieval.

This system was written in Cap'n Software Version 1.7. Versions for other than Apple II are being developed.

For availability contact:
The Commitree Group
Box 14431
San Francisco, CA 94119
or call the original Tree:
526-7733.


## FORTH, INC. NEWS

## MAJOR EXPANSION PLANS

FORTH, Inc. is now entering a major expansion phase, according to President Elizabeth Rather. Appearing on a panel on "Programming Languages for Small Systems" at the recent NCC in Chicago, Rather observed, "The level of excitement and enthusiasm about FORTH in the industry is tremendouus. We are increasing our number of OEM'S and we have been approached by several major silicon manufacturers desiring to obtain marketing rights for special versions of polyFORTH. Arrangements are also being made to produce the FORTH processor, and we expect this project to start very soon."

## LIFEBOAT REPRESENTATIVE VISITS

Masa Tasaki, Managing Director of Lifeboat, Inc., FORTH, Inc.'s distributor in Japan, spent two days at FORTH, Inc. recently to discuss mutual marketing plans. Lifeboat, Inc. is one of the few software distributors in Japan, and polyFORTH is the top of their product line. Tasaki has installed over 40 polyFORTH systems in Japan in the past year, and plans to sell an additional 50 polyFORTH systems by the end of 1981.

## STARTING FORTH BOOK PREPRINTS AVAILABLE

STARTING FORTH, a 380 -page book introducing the FORTH language and operating system will be published by Prentice-Hall this September in both hard and soft-bound editions. FORTH, Inc. is offering limited preprints to customers until then. The preprint, numbered and signed by both author Leo Brodie and Charles H. Moore sells for $\$ 50.00$ (plus $6 \%$ sales tax for residents of California). You may reserve a copy of STARTING FORTH by calling Winnie Shows at (213)372-8493. All orders must be pre-paid.

RECENT FORTH COMMERCIAL APPLICATIONS
Work has just been completed for Raytheon Corporation on a terminal cluster (up to 32 terminals with a single concentrator). Each component of the system is controlled by an 8085 processor, and all are programmed independently, using polyFORTH. This is a capability they've never had before -- to do custom programming and provide extensibility. Terminals up to two miles away can be polled at a rate 30 times faster than the previous protocol, which was written in assembler. Dean Sanderson was the principal programmer on the project.

The famous 200" Hale Telescope at Mt. Palomar Observatory (near San Diego) has recently installed a polyFORTH system for data acquisition and analysis using a PDP11/44 and a Grinnell display processor. The Observatory has been using FORTH since the early 1970 's, including a miniFORTH system installed in 1975 and an early polyFORTH installed in the late 70's. Barbara Zimerman, a programmer at Cal Tech (which operates the observatory) said, "I am extremely impressed by the level of polish and sophistication in polyFORTH, and the performance of this system is outstanding." The type of work done involves reading data from an 800 x 800 array of CCD sensors, integrating and recording the data, and displaying it in the Grinnell. Charles Moore installed the system, which features a comprehensive math package for analysis as well as basic image-processing functions.

A by-product of this installation is the availability of polyFORTH in RKOS disk cartridges. These are available with onsite installation.

SCHEDULE OF UPCOMING FORTH, INC. SEMINARS AND WORKSHOPS:

| Location | Seminar | Workshop |  |
| :--- | :--- | :--- | :--- |
|  |  |  |  |
| Palo Alto | June 4 | June | 5 |
| Houston | July 7 | July | 8 |
| Tampa | July 9 | July | 8 |
| Irvine | July 23 | July 24 |  |

# PARAMETER PASSING TO DOES 

David McKibbin<br>Sygnetron<br>2103 Greenspring Drive Timonium, MD 21093

Often in programming one runs into the case where several different processes share similar structures. Not wanting to waste time or space for redundant code, the programmer usually creates a subroutine or procedure to execute the basic structure. Then the individual processes merely pass arguments to the prodecure to accomplish their task. Several schemes can be used to pass these parameters. In simple cases, the stack can be used directly. This is the typical act of programing in FORTH.

## : DELAY 0 DO LOOP ; (SPIN POR A KHIIE )

100 DEIAY ( COUNT PASSED ON THE SNACX)

However, as the procedures get more complex it gets more and more difficult to keep track of the passed parameters especially when the procedure itself is using the stack heavily. Also many times it is necessary to pass not only numbers but operators or words as parameters. One means of accomplishing this is via <BUILDS DOES $>$. Parameters will be stored in the parameter field of the newly defined word and accessed from DOES) via a new word \{\$\}. 1 \$ will push the first parameter on the stack, 2 \$ will push the second, etc. All parameters are 16 bits. Variable R\# is used to store the parameter base address.
: ₹ 1 - DUP + R ${ }^{\prime}$ @ + @ ; ( PUSH THE N'TH PARAMEIER )

EXAMPLE ZZZ 90 , ' EMIT CFA, ( TYPE A 'Z'')
EXAMPIE SPC 10 , ' SPACES CEA , (TYPE 10 SPACES )

Now that the mechanics are explained the following example will more fully demon-
strate its usage. Both DUNP (16 bit dump) and CDUMP (8 bit dump) share a common structure with only a few inner words differing. DUMPS is a new defining word used as a procedure for both DJMP and CDUMP.

```
: U.R O SNAP D.R ;
```



```
ADDRESS )
            BASE @ R>HEX ( SAVE BASE AND SEI HEX )
            ONER + SNMP ( Convere TO begmNmg
    ALD END ADLRESS )
        BEGN
            CR IUP 4 U.R 2 SPACSS ( TMEE ADDRESS)
            1 $0 m
                DUP 2 $ EmOUIE 3$ U.R 4$ +
                ZWP = OVER 16 MDD O= OR IF
                    lrave timen
            LOP
            2WUP = THORMINL QR
        UNTIL
    DROP DROP CR
    R\ BASE ! ; ( ELSIORE RASE )
DIMES CUMP 16, 'CO GFA , 4, 1,
    ( 2-Almiess, 1-counir )
DLMPS DaP 8, ' CCFA , 6, 2,
    ( 2-ATDRESS, 1-COUNI )
```

What has been accomplished is akin to passing procedures/functions as parameters in Pascal. I expect that there are other ways to do this FORTH beyond what has been proposed.

## FIG-FORTH UNDER OS-65U

Software Consultants has announced the availability of Fig-FORTH under OS-65U for the Ohio Scientific Line. The package includes assembler and a terminal oriented editor and is available now for $\$ 79.95$.

This version is said to support harddisk, multi-user systems and may even be run in one partition and BASIC in another.

For more information contact:

Software Consultants
7053 Rose Trail
Memphis, TN 38134
(901) 377-3503

# COMPILER SECURITY 

George W. Shaw III SHAW LABS, LTD.<br>17453 Via Valencia San Lorenzo, CA 94580

How it Works and How it Doesn't (Adapted from a section of the Acropolis A-FORTH manual)

There is much argument about parameter validation and error detection in FORTH. Many problems exist with many good solutions. Fig-FORTH and its derivitives have taken one route of extensive protection in compiler directives and their associated words. This is not an only solution in this area. Its extensiveness may not be necessary. There may be better alternatives. Read on, learn how fig-FORTH works, consider the options and then decide. Your opinion and ideas are needed.

Fig-FORTH and its derivitives provide a type of compiler error detection referred to as "compiler security". Compiler security provides protection against structural programing errors made by the programer as well as insuring the proper machine state and, in a very few instances, the validity of parameters. Though it depends on the type of programming, the most common errors are structural errors*, machine state errors, and then parameter errors, respectively.
(* structural errors may be caught internally by detecting parameter errors. See text.)

## STRUCTURAL ERRORS

The compiler security system uses two methods to trap structural programing errors inside of colon-definitions. Structural errors are those caused by incorrect program structure; either improper nesting of structures or not completing a structure inside of a definition. Either of these conditions would cause the program to compile incorrectly
and could cause disastrous effects (i.e. a system crash) at run-time. The methods used by the compiler security system entail either checking a value on the top of the stack (to verify the proper nesting of structures) or checking that the stack position is the same at the end of a definition as it was at the beginning of the definition (to ensure program structure completion). These two methods probably trap about ninety percent (90\%) of the structural programming errors that a programmer might make.

The first in each of the paired structural compiler directives (i.e. pairs such as IF THEN, DO LOOP, etc.) leave on the stack at compile time a value which is checked by the ending structure to ensure the proper nesting of structures. For example the word IF leaves, in addition to the other data necessary to compile an $I F$, the value of two (2) on the top of the stack. The words ELSE and THEN remove a value from the top of the stack and check to see if it is a two (2). If the value on the stack was not a two (2), a Conditionals Not Paired error (\#19) results, and compilation is terminated (control returns to the keyboard). If the value is a two (2) the remainder of ELSE or THEN executes, removing the necessary data from the stack to finish the structure, and compilation continues on to the next word.

Below is a table of the conditional pairs for the current structural compiler directives, with the values placed on the stack open and the values removed from the stack in parenthesis. Note that UNTIL and END as well as THEN and ENDIF have the same effect. Only the former of each pair are presented here for clarity.

| BEGIN | 1 | UNTIL | (1) |  |  |  |
| :--- | :--- | :--- | :---: | :--- | :--- | :--- |
| BEGIN | 1 | WHILE | 4 | REPEAT | (1) (4) |  |
| BEGIN | 1 | AGAIN | $(1)$ |  |  |  |
| IF 2 |  | THEN | (2) |  |  |  |
| IF | 2 |  | ELSE | (2) 2 | THEN (2) |  |
| DO | 3 |  | LOOP | (3) |  |  |
| DO | 3 |  | +LOOP | (3) |  |  |
| DO | 3 |  | /LOOP | (3) |  |  |
| DO 3 |  | +/LOOP (3) |  |  |  |  |

Note that ELSE tests and replaces the same value on the stack. Because of this the current compiler security system cannot detect the presence of multiple ELSEs in a definition. For example, in the definition:
: ELSE-EXAMPIE flag - true or false message ) IF ." True part 1 " EISE ." False part 1 " ELSE ." False part 2 " ELSE :"False part 3 " THEN ;
if compiled, (and it will compile,) and then executed with a boolean value (zero or non-zero) on the stack, will execute without crashing the system. But the execution may not be what you expected. If entered with a true flag (non-zero) the "True Part 1" and the "False Part 2" will print, while if entered with a false flag (zero) the "False Part 1" and "False part $3^{\prime \prime}$ messages will print. To borrow a phrase from Kim Harris, probably "Not what you had in mind!".

This is the only case $I$ know of where the compiler security system plainly does not work, but there are probably more.

How is this, apparently incomplete, structure checking performed? Read on.

The values on the stack is verified by ?PAIRS . For example the words ?PAIRS , BEGIN and AGAIN are defined as follows:

```
: ?PAIRS - 19 ?ERROR ;
: BEGIN ?COMP HERE 1 ; IMMEDIATE
: AGAIN 1 ?PAIRS COMPILE BRANCH BACK
; IMMEDIATE
```

BEGIN first checks to make sure that it is being executed in compile mode (inside a definition) with ?COMP which issues an error if it is not. It leaves the current dictionary address on the stack (HERE) as a branching reference for AGAIN, and then the 1 as the first of a conditional pair. When AGAIN later executes during the compilation of the definition it first checks the stack to see that a BEGIN preceded it at the same level of nesting by executing ?PAIRS . ?PAIRS expects to find a matched pair of values, in this case ones (1), as a matched set of condi-
tional pairs. If ?PAIRS does not find a matched set, it aborts with a Conditionals Not Paired error ( ${ }^{(19)}$ ). If the values on the stack are paired, it removes them and returns.

The above simple form of error checking is very effective, but as structures become more complex, manipulating and maintaining the stack values can become cumbersome and unwieldy. The above is also not yet complete. One more check must be executed to ensure that the structures in the definition have been completed. Since the above error checking leaves data on the stack if a structure has not been completed, the simplest check is that of the stack position. When a definition is entered $:$ (colon) stores the Current Stack Position in the user variable CSP . At the end of a definition, ; (semi-colon) executes ?CSP to compare the current stack position to the value stored in CSP. If the values differ a Definition Not Finished error (\#20) occurs indicating that either data was left on the stack or that too much data was removed from the stack, i.e. that a programming structure was probably not completed. The word "probably" is used here because other conditions, such as the improper or sometimes various proper uses of the word LITERAL, will cause the same error condition to occur.

## MACHINE STATE ERRORS

The loading and execution of a FORTH program causes the system to enter several different machine states. Three of these are loading, compiling, and executing. Each of these states is defined by its own set of parameters and some states may even overlap. For example, while loading a screen off the disk the machine will be either executing or compiling. Here the loading state has overlapped with either the execution or compilation state. The machine cannot be in the execution state and the compilation state at the same time, though the states may be interleaved. An example of interleaved states is the use inside a definition of a program segment similar to this:

## [screxn 3co] + Literal

which temporarily suspends compilation to calculate the value within the brackets and then compiles it as a sixteen (16) bit literal. Remember though, that to compile, the machine is executing a program, and that compiler directives (such as LITERAL above) execute during compilation to perform their task, but the machine state remains that of compilation.

Certain words require that the machine be in a specific tate to execute properly. These words are programmed to contain one of the following vords:

## ?COMP ?EXRC ?LOADING

which check for their corresponding state and issue an error message if the machine is not in that state. Below is a description of each of the above words and the parameters which deterwine the current machine state.
?EXEC or ?COMP
The execution state or compilation state is determined by the value of the user variable STATE which has a zero ( 0 ) value if the machine is in the execution state and a non-zero value the machine is in the compilation state.

## ?LOADING

Loading is determined when the value of the user variable BLK has a non-zero value. A value of zero for BLR indicates that input is coming from the user's terminal and that the machine is therefore not loading.

Th : above words are defined as follows:

```
: ?EXEC STATE @ 18 ?ERROR ;
: ?COMP STATE @ 0= 17 ?ERROR ;
: ?LOADING BLK & 0= 22 ?ERROR ;
```

If the machine is not in the execution state when ?EXEC executes an Execution Only error ( ${ }^{(18)}$ ) occurs.

If the machine is not in the compilation
state when ?COMP executes a Compilation Only, Use in Definition error (\#17) occurs.

If the machine is not in a state of loading when ?LOADING is executed a Use Only When Loading error ( $\mathbf{K}^{\boldsymbol{W}} 22$ ) occurs.

The testing of machine states as above is necessary when words such as BEGIN and AGAIN (see example in STRUCTURE ERRORS above) are used. These words may only be compiled because they must compile something other than themselves which is not known at the time they are executed.

## PARAMETER ERRORS

During compiling and similar operations there are only a few parameters which are actually checked. In most cases, the parameters checked are those involved in the other areas of compiler security or those which deal with the size or validity of the dictionary and stack.

The words involved in other compiler security areas are ICSP, ?CSP, ?PAIRS . These words are used to protect against structural programming errors ad described above in STRUCTURAL ERRORS. An explanation of each of the uses of these words is as follows:
! CSP ? CSP
These words are used together to check for changes in the stack position. ICSP stores the current stack position in the user variable CSP . ?CSP compares the value in CSP to the current stack position and, if they are not the same, issues a Definition Not Finished error ( 20 ). !CSP and ?CSP are currently used in : and ; respectively to ensure that all structures in the definition have been completed before the semi-colon. Any structures uncompleted will leave data on the stack and thus allow ?CSP to flag the error. These words can also be used to check the stack effect of user definitions. For example, if a definition should have no stack effect (leaves the same number of items on the stack as it removes) the following would test this:

## !CSP cecc $\quad$ CSP

wich would execute a definition named sccc and issue a Definition Not Finished error ( $\$ 20$ ) if the number of items on the stack at the beginning and end of the definition were different.
?PAIRS
This word is used when testing for eorrect structure in compiler directives see STRUCTURE ERRORS) to check that the value of the two numbers on the stack is -he same. If the value of the two compiation conditionals on the stack is not -he same, a Conditionals Not Paired error 419) occurs. ?PAIRS can be used to test similar situations in user programs, but :he error message given will be the same error \#19).

The checks on the dictionary and stack consist of testing the stack for underslow, the dictionary and stack for overslow, and the name of the dictionary entry to be created for uniqueness ( in A-FORTH this test is optional and there is a test to ensure that a definition name is not null). Some of the tests are performed during the execution of other functions by the testing word (such as the tests performed by WORD and by CREATE ). Only the testing performed by these words will be described here.

## CREATE

This word creates a dictionary header for a new word. In the process of creating this header a dictionary search is performed to check that the header is unique. The message given if a duplicate is found is Isn't Unique (解). This is not a fatal error but just a warning. A-FORTH allows the disabling of this test (and the associated message) and performs another test for a dictionary entry whose name is a null. The creation of a dictionary entry with a null name is not allowed because the null is the name of the entry interpreted at the end of the disk or terminal buffers. If an attempt to create a null entry is detected a Null

Definition Name! error ( $\ddagger 9$ ) is given. If a dictionary entry with a null name were created, the system would attempt to interpret this as the end of the current buffer with umpredictable results.

## ?STACK

This word checks that the parameter stack is within bounds. It compares the current stack position (by executing SP@) against the base stack position in user variable $S 0$ to check for a stack underflow. It also checks that there are at least 128 bytes of dictionary space left (to leave room for PAD and stack work). If the stack underflows an Empty Stack error (事1) is given. If the stack comes within the 128 bytes of the dictionary a Full Stack error (\$7) is given. ?STACK is not executed at runtime unless compiled by the programer, though it is executed frequently during compiling and text interpretation.

## WORD (A-PORTH only)

This definition moves text from the current input buffer to the head of the dictionary. The error test performed checks that there is enough space between the head of the dictionary and the top of the stack for the text about to be moved. If there is not enough space a Dictionary Full error (*2) is given. This prevents the system from crashing by writing over its own stacks.

## DO WE NEED IT?

Should we have all this security all the time? Or just when we think we need it? Fig-FORTH currently does not give us a choice on the matter. Sure, we can compile on top a new set of compiler directives which don't have the tests, but we have then already wasted all the memory for the secure directives, the ?XXX words, and the lot. The reverse course I consider more appropriate. The kernel system should have as little protection as possible. The system should not suffer the overhead for those who do not desire it. If security is desired, a "Novice Programmer Protection" package could be
compiled into aser's area which would include all the words necessary to protect him or her (and the other users) from him or herself. This would allow protection even for the words such as ! (store), FILL and CMOVE when desired.

Something as simple and extremely effective as the !CSP and ?CSP in : and ; respectively may be left in the kernel system to give warning to even the best of us when necessary. Definitely, also the stack checks at compile time and possibly the uniqueness (though it should be optional) and null definition (currently A-FORTH only) checks should be left in, but the structure and state testing is ften incomplete and annoying. Anyone who has tried to write and secure a good general CASE structure, or a BEGIN WHILE REPEAT loop which allows multiple WHILEs will know what a pain it is to try to secure them in a reasonably complete fashion. For these people compiler security dosn't work. Additiona!ly, new structures transported from my system to another may not remain secure because the same conditional pair numbers used in my structure on my system may have been used in a different structure on the other system. Again, the compiler security dosn't work.

The same method used in high level structure testing is also used in one known assembler, which the author considers totally inappropriate. If one is programming in FORTH assembler one is doing so for speed, which may require not being structured at all.

Currently, the matter of compiler security is being studied by the group writing the next 8080 fig-FORTH version (which could possibly outline a new model). Should we have all the protection all the time, or just some of it and a programmer protection package? Or maybe there is a better alternative. Your input is wanted and needed. Write to the 8080 group at FIG, PO Box : 105, San Carlos CA 94070 and tell as what you think.

## NEW PRODUCTS

## POLYMORPHIC FORTH NOW AVAILABLE

FORTH is now available for the polyMorphic Systems SSSD 5" systems (8813 \& 8810). The PolyMorphic disk operating system has been patched in and the system is interfaced to the PolyMorphic operating system. PolyMorphic FORTH includes a modified systems disk, and brief documentation on changes to interface to the PolyMorphic SSSD 5" disk operating system -- based on 8080 Fig-FORTH. Price is $\$ 50.00$. For more information contact:

> Ralph E. Kenyon, Jr.
> ABSTRACT SYSTEMS, ETC. 145-103 S. Budding Avenue Virginia Beach, VA 23452

## FORTH FOR HP83/HP85

A disk based FORTH is now available for the HP85/HP83 personal computers. The implementation is the FIG FORTH 1978 standard with some machine dependent utilities. User receives both 16 k and 32 k versions with user space being $2 k$ and $18 k$ respectively. Both versions require a disk. Included is an assembler, a FORTH decompiler and editor. This is not an HP supported product but available through the user's library. FORTH, in object form (no source), an assembler, decompiler and editor, in source, are sent on a disk. This product recommended for experienced users only! Those familiar with FORTH should have no trouble using this system (i.e. there is no manual inluded). However, sufficient references are given. Current cost is $\$ 50.00$. For more information contact Nany Reddington at (503) 757-3003.

## FORTH PROGRAMMER AVAILABLE

3 mos. experience with FORTH (also know BASIC \& COBOL) Active member of F.I.G. Contact: Martin Schaaf, PO Box 1001, Daly City, CA 94017 (415)992-4784

# USERSTACK 

Peter H. Helmers<br>University of Rochester Department of Radiology Medical Center, Box 648 Rochester, NY 14642

## INTRODUCTION

One of the advantages of FORTH is its use of a stack oriented architecture. In conventional FORTH implementations, one has available two kinds of stacks: the return stack and the parameter stack. In general, the return stack is used to keep track, at execution time, of the path of invocation of nested FORTH words while the parameter stack is used to manipulate data used within and/or passed between FORTH words.

Unfortunately, in the real world, such a clean segmentation between parameter data and execution nesting data tends to break down. For example, DO...LOOPs are implemented by using the return stack to keep track of the loop count and associated data. The motivation for this violation of the sanctity of the return stack with DO...LOOP parameters is the desire to separate the DO...LOOP data from any parameters being used by the programmer within the loop. Failure to do so would allow confusion of loop parameters with actual user data -- causing a consequent abnormal execution of the DO...LOOP arising from an unwarranted modification of loop parameters.

In addition to the above saving of DO...LOOP parameters on the return stack, it is not uncommon practice for a programmer to want to save some parameter stack data in order to be able to first calcuiate using data beneath it. One previously employed method to do this was to temporarily push the parameter stack data onto the return stack, and then later

Editor's Wote: Mr. Helmers uses URTH, a dialect of FORTH.
retrieve it when subsequently needed. Admittedly, this is an easy - lazy! -way to achieve tranisent data storage. But woe unto those who forgot to pop the return stack of this temporary data...!

## USER STACKS

The "user" stack concept allows a FORTH program to retain the convenience of an auxiliary stack, but in such a way as to avoid mixing temporary data with execution time return information. As an added convenience, this concept allows creation of multiple, named, stacks which can be typed according to the number of (two byte) words per stack element.

A user stack can be thought of as an array (integer, double precision, or real) of data which has implicit addressing. Consider, by way of analogy, a conventional array such as:

100 ()DIM MY-ARRAY
One would store the 53rd integer element by explicitly stating the index:

## 52 MY-ARRAY ! ( ZERO ORIGIN...)

This would take data from the top of the parameter stack and store it in MYARRAY. Alternatively, one would access an integer from this array by:

## 27 MY-ARRAY ©

The disadvantage of arrays is that they require both an explicit index, and an explicit load (0) or store (!) operator. While an array could be used for temporary storage of parameter stack data, such programming practice is not neccessarily clear or efficient.

So how does a user stack help us? Consider the integer user stack defined:

100 STACR MY-STACK
MY-STACR would, in this case, have a size of 100 integer elements. Data can be put into this user stack from the top of the parameter stack by:

## PUSH MY-STACR

while it can be retrieved back to the parameter stack by:

## POP MY-STACK

Note that addressing is implicit-- there are no indices - and that the direction of data transfer is set by the PUSK and POP words.

## USER STACK WORDS:

In practice, three types of user stacks have proved useful; STACK, DSTACK, and FSTACK. While stack variables created by these three defining words all use the PUSH and POP words to save and retrieve data, the amount -- or type -- of data pushed or popped differs. As discussed earlier, STACK deals with integer (two byte) words. DSTACR consists of elements of double precision integer words (four bytes) while FSTACK elements are floating point numbers (six bytes). All three of these words are defined in terms of an arbitrary $n$-precision NSTACK word which allows specification of any number of two byte words per stack element.

Two other words are also useful with user stacks. There are EMPTY-STACR and ?STACR. Note that both of these cannot (presently) be used within colon definitions. The line:

## EMPTY-STACK MY-STACK

will, for example, reset the stack pointer for the user stack: MY-STACR so that it will be empty. Again using the MX-STACK example,

> ?STACK MY-STACR
will dump out the contents of the stack from the top of the stack through the bottom of the stack. ?STACK is intended purely as an aid in debugging.

## IMPLEMENTATION:

As was previously mentioned, STACR, DSTACK, and FSTACR are all defined in
terms of a more general NSTACK defining word. A line such as:

## 224 NSTACR WIDE-STACR

will define a 22 element stack with eight bytes (four words) per element. NSTACK has two primary parts. The first part, executed when a new stack is defined, builds a FORTH word header, stores some stack definition parameters into the dictionary, and finally allocates the actual dictionary space for the stack, The second part, written in 8080 assembly language for speed, defines the execution time actions taken by the stack variable. Both of these defining parts will be explored in greater detail below.

The format of the user stack in the dictionary is shown in Fig. I. It consists of a normal FORTH header, followed by the following four stack definition parameters:
a) current stack pointer (two bytes)
b) *words per stack element (one byte)
c) maximum stack pointer address (two bytes)
d) minimum stack pointer address (two bytes)
*BYTES FIELD
COMMENTS


* characters saved for word neme

Points to ;000 pert of ISTACK def.
\# bytes needed to contain specified \# of stack elements

Figure 1 -- Dictionary Layout for a Stack Type Variable

```
* macx - 1so
    stack data tymes -- moe 23 0cl mo 
```



```
        CMLEO MOW WITH TNE OIVEM MUNSR OF ELEMENTS., STACK DATA,
```



```
        BTack)
```



```
        not C. (monos/stx ficm). (mar eo mem, )
```




```
        ~ LMLD. N DCX, DN NON, NMCX; EM Mov. mPNED. I EN IP
        IF. i mop mata from netack to man+ettim erack;
```



```
            ; ca+ IN EC!
    -->
*** Lock - 131
    1 matmax defimitiom cowet
        #EOIN., ' NOVImB DATA. wome AT A TIME, men nums.
            LDNX. I INM. A E MOV. E LNAX. INX, a o MON.
```



```
            a nc: O=
```






```
            MSU mOp. A DCR. o=,
        EMD
    C M MON. H INY, B M MON. I EANE MEN COP:
    RO HLD. H E MOV. H iNX. H D MON. H IMX, W ENLD. MEXT MP.
** H_OCX © 152
    ( stacx TVAE Malamese cowrt)
    coos mor O w ixi, muwn me.
    11. mumy now.
    SHACK I MBTACK: & IMTEOSN ELM,MTE
```



```
    fstack a mstack : FlomyImo motwy klomente)
```




```
    EmTY-8tmck j'{ dre 3 * Simp:,
```




```
    & Elat 2mm, to uen s,ack e
    18
en
OK 100 STACK MY-STACK
OK 35 STACK YOUR-STACK
OK 1: 22 33 44 55 66 77 90 99
OK PUSH MY-STACK
OK PUSH MY-STACK
OK PUSH YOUR-STACK
OK PUSH MY-STACK
OK. PUSH MY-STACK
SK PUSH MY-STACK
OK PUSH YOUR-STACK
OK + POP MY-STACK -
-11 OK
OK POP YOUR-STACK POP YOUR-STACK 2DUP . . + .
77 33 110 OK
OK POP MY-STACK.
    S5 OK
OK POP MY-STACK .
\infty OK
OK POP MY-STACK .
88 OK
OK POP MY-STACK.
99 OK
OK
OK
OK
ON
OK PSTACK MY-STACK
USER STACK EMPTY OK
OK 11 PUSH MY-STACK 22 PUSM MY-STACK TSTACK MY-STACK
3A7A 0016 000B
O*
OK EMPTY-STACK MY-STACK PSTACK MY-STACK
USER STACK EMPTY OK
O
O*
```

Note that the stack, consistant with the 8080 architecture, grows down in memory. Following these stack parameters is the actual stack area which is allocated in the dictionary.

The PUSH and POP words are code definitions (for speed) which push a 0 or -1 flag value to the top of the parameter stack. Thus, when the stack variable is subsequently executed, this flag is used to differentiate between popping from the user stack ( $f 1 a g=0$ ) and pushing to the user stack (flag=1). The assembly code is thus separated into two very similar execution loops which move stack data one word at a time until the proper number of words for the stack element have been moved; these two loops differ only in the direction of the data transfer. In both loops, the $A$ register contains the current word count which is intially set to the number of words per stack element and decremented each time through the loop. The BC register pair contains the current user stack pointer while the HL register pair contains the address of the stacks parameter field so that the new user stack pointer value may be saved after all words within the stack element have been transferred.

## CONCLUDING REMARRS

These user stacks have been optimized to provide rapid execution speed at the expense of high level transportability and error checking for a stack pointer out of bounds. It is felt that the concept, in whatever realization, is important since it provides a very readable and structured method to temporarily store and sort data without having to resort to such "unclean" practices as using either explicitly addressed arrays or the return stack. It's the type of FORTH word that, once you have it, prompts the question: "it's so obvious, why didn't someone think of it before?"

## NEW PRODUCT

## STAND－ALONE FIG－FORTH FOR OSI

FORTH Tools has announced stand－alone Fig－FORTH for all OSI mini－floppy com－ puters that combines Fig－FORTH with stand－ alone machine drivers by FORTH Tools． With this system OSI－65D is superfluous－ with FORTH booting up directly，yet the disk is OS－65D compatible．

Since FORTH Tools FORTH dispenses with the OSI operating system，FORTH Tools has developed disk，display and keyboard drivers for the OSI hardware．

FORTH Tools FORTH for OSI is strictly compatible with Fig－FORTH．All words in the Fig model，including disk support， work correctly．Portability to other machines is also claimed．

Stand－alone Fig－FORTH for OSI is avail－ able on one 5－1／4＇disk for Cl（Super－ board），C2 and $C 4$ machines with 24 K ． Product includes a structured 6502 macro－ assembler and disk utilities designed by FORTH Tools and the FIG portable line editor．Complete technical documentation and the fig－FORTH glossary are also included．The complete price is $\$ 49.95$. For more information contact：

FORTH Tools
Box 12054
Seattle，WA 98102

```
2EA- EDI---
# AV AFR--- OHA- T!E \ET--- IN =HE LNS- ISS-- iSO--
OR-- \NC- USI-- ONL- IHR-- LET--- NAM- FIE--- HAS
```






```
KEE---- LEG-------
WE STI-- DON-- SEE THE NEE- FOR 31 CHA------ NAM--
* OHE GEN---- adS-
シつ-- エズー-
OMNO
CHO-- NOO--
FOR-- INC-
D.S- MR. ERE- THO---- IS NOT AN EMP----- OF
    FOR-- INC-
```


# A STACK DIAGRAM UTILITY 

Barry A．Cole
3450 Sawtelle Blvd．\＃332
Los Angeles，CA 90066

## INTRODUCTION AND CONCEPT

A year and a half ago，when $I$ was still fairly new to FORTH，I spent a lot of time drawing pictures of stacks as $I$ made up programs．I crumpled them up and started over each time $I$ changed them．As sections were debugged，I drew up another copy to document the code．When $I$ found an error，I would have to redraw whole series of stacks，just as a cartoonist would have to change a whole series of frames．It soon became clear that I was expending time to do rather tedious work．I came up with an idea for an automated tool to update these diagrams． I thought up a way to represent the stack data easily and an approach to implement the tool．The original implementation was done in 8080 polyFORTH by my co－worker Greg Toussaint．We collaborated in the initial debugging and then passed it back and forth over the next four months． After nearly a year in active use，I converted it to fig－FORTH and updated several messy areas to be more straight－ forward．The results of these pursuits are detailed in this paper for more general consumption．

## ORIGINAL IMPLEMENTATION

The original program was going to take push and pop information from the keyboard to generate pictures of what was on the stack．It became immediately clear that the stack could more easily be represented horizontally than down the page．We chose to put the stack to the right so that the size of the stack could be read like a bar graph．I figured that if I represented each item on the stack as an address pointing to a count and printable string， that many of the stack diagram words would be identical to the FORTH word equiva－ lent．Thus，DUP，OVER，DROP as well as many other primitives would be coded before $I$ started．Even as it was being
built, the tool grew to get the source codes directly from disk and then to generate a printer format spool file also onto FORTH screens. Keeping track of values when an $I F$ was encountered and restoring them on ELSE and THEN was added. This generates a warning message if the two paths leave different numbers of parameters on the stack. Finally, concatenation of strings for algebraic and logical expressions was added.

## USAGE AND OPERATION

The main routines called by a user are:
screen $\# \mathrm{DOC}$ defname screen SDOC screen\# PRIDOC PDOC
to document 1 definition to document a whole screen to print fram given screen to print last documentation

The program clears the display stack before each colon definition. A search is made for the first colon on SDOC or the specified name following a colon on DOC. The name of the function is displayed along with the currently empty stack contents. It requires user input to continue since the entry conditions of the routine are unknown. It prompts "DROP?" to see how many excess elements should be dropped from the stack, A carriage return suffices to leave it alone. It continues with the prompt, "PUSH VALUE?". For each symbolic name of a value on the stack, a free form name should be typed followed by a carriage return, The prompt will be repeated until a line consisting of only a carriage return is typed. There are no limitations imposed on the input, however, it is advised that nulls and tabs should not be included as this will detract from the clarity of the final output. The ? cogram will then continue reading words from the source screen and generating output lines to the console and spool file.

In a typical sequence, up to about a dozen lines will be handled without intervention. For example, occurences of DUP, DROP, and numeric literals will be processed automatically. When a @ is encoun-
tered, it will revert to the prompts since it is not known what a symbolically appropriate name is for the fetched value.

Processing will terminate with an "OR" for sucessful completion of the screen or colon for SDOC or PDOC, respectively. If stack underflow occures, it will abort. It is good practice to do a PORTH after an abort condition to insure that the stack vocabulary is properly exited. A user abort is also provided. This is accomplished by typing an escape key followed by a carriage return in response to the "PUSH VALUE?" prompt.

## SAMPLE DIALOG

The package creates a special stack vocabulary as well as the user entry points. The use of the package is best seen by example. Figure 1 is a sample dialog. Notice how little intervention is required and how the ELSE restores the stack values. Figure 2 is the source that was used in the examples. Figure 3 is the printer output as displayed by PDOC.

FIGURE 1

| 100 SDOC |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| ANALYZE | 1 |  |  |  |
| DROP? |  |  |  |  |
| PUSH VALUE? | addr |  |  |  |
| PUSH VALUE? | len |  |  |  |
| PUSH VALUE? |  |  |  |  |
| analyze | 1 addx | len |  |  |
| SWAP | \| len | addr |  |  |
| INCH |  |  |  |  |
| DROP? |  |  |  |  |
| PUSa Value? | char |  |  |  |
| puSh Value? |  |  |  |  |
| INCH | len | addr | char |  |
| DUP | len | addr | char | chas |
| 7 F | 1 len | addr | char | chay 7F |
| - | 1 len | addr | char | (char-7p) |
| IF | 1 len | addr | char |  |
| DUP | l len | addr | char | char |
| OD | 1 len | addr | char | char OD |
| - | 1 len | addr | chay | (char-0D) |
| $1{ }_{\text {If }}$ | \| len | addr | chay | char |
| OUCH |  |  |  |  |
| DROP? 1 |  |  |  |  |
| PUSH VALUE? |  |  |  |  |

FIGURE 1 (cont.)


FIGURE 2


FIGURE 3

PDOC ETACK DIAGRAN - SCREER 100

| nualyse | 1 | addr | 1en |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| SWAP | 1 | len | addr |  |  |
| 1 IVCH | 1 | Ien | addx | chat |  |
| D09 | 1 | 1en | addz | char | char |
| 77 | 1 | 1en | add: | chat | char 7 F |
| - | 1 | len | addz | chas | (chat-7F) |
| 17 | 1 | len | addi | chat |  |
| DUP | 1 | len | adds | chat | chat |
| OD | 1 | Ien | mddt | char | chat OD |
| - | I | $1 e n$ | addz | char | (chat-OD) |
| 19 | 1 | Ien | eddt | char |  |
| DUP | 1 | len | addr | char | chas |
| OUCH | 1 | len | addt | char |  |
| over | 1 | len | eddr | chat | addz |
| C! | 1 | 1 1en | addi |  |  |
| 1* | 1 | 1en | (addt | t+1) |  |
| SWAP | 1 | ( ${ }^{0}$ ) | +1) | Ien |  |
| 1- |  | (add | +1) | (len- |  |
| ELSE | 1 | 1 en | addr | char |  |
| ( | 1 | len | addy | chat |  |
| DROP | 1 | 1en | eddr |  |  |
| Sutap | 1 | addz | len |  |  |
| DROP | 1 | addt |  |  |  |
| 20 | 1 | add: | 20 |  |  |
| ODCA | 1 | -1dt |  |  |  |
| 0 | 1 | addr | 0 |  |  |
| OVER | 1 | eddr | 0 | addr |  |
| CI | 1 | sddt |  |  |  |
| 0 | 1 | medr | 0 |  |  |
| THEX | 1 | dert | 0 |  |  |
| ELSE | 1 | 1 m | eddy | chas |  |
| 1 | 1 | 1 len | addr | chas |  |
| Drop | I | 1en | add: |  |  |
| 8 | 1 | 1 n | eddt | 8 |  |
| OUCa | 1 | $10 n$ | add: |  |  |
| 1- | 1 | 1 en | (addt | c-1) |  |
| gmap | 1 | (sad | -1) | 1 n |  |
| 14 | 1 | (a) | (-1) | (1ent |  |
| cimen | 1 | (a) | (-1) | (1ent |  |
| ${ }_{0}^{1}$ | I | ladd | (-1) | (1ent |  |

$3 n \mathrm{~m}$
$n$

## CONDITIONALS

The IF...ELSE...THEN construct automatically saves and restores the stack values. A mismatch in number along the two paths produces a warning message,
"STK ERROR, ELSE -m THEN -n"
where $m$ is the number of parameters left on the stack at the end of the IF clause and $n$ is the number left when the THEN is encountered. The DROP/PUSH prompts are presented for the user to attempt recovery. A known cause of this message is a -DUP preceding the IF, as this is not handled.

## SPOOLING TO DISK

To be useful, a hard copy of the output without all the intermediate operator conversation is useful. It is also quite possible that a machine readable version would be handy to facilitate distribution of the documentations. A spool file is generated to satisfy these requirements. $\therefore$ may later be displayed or printed by ?DOC.

The spool file contains the encoded screen from which the diagram was made followed by variable length lines separated by carriage return characters. The file is terminated by an ascii null sharacter. It resides on a set of consecutive screens. The first screen and a aximum number are determined by literals $\therefore$ SPIT and PRTDOC. I use 10 screens starting at 230. These may be copied Esewhere and printed by PRTDOC. Failure :o copy them will cause the listing to be ist the next time a function or screen is i i grammed.

## - YPLEMENTATION PROBLEMS AND SOLUTIONS

It is important not to search the standard vocabularies when diagraming stacks. This is because actions are different for the same name, depending i? =perator + must concatenate the symbolic rame strings representing these elements
with an embedded plus sign, rather than adding the top two elements on the stack. Also, not all operators are defined. On detection of this case, the diagrammer must shift control to the operator prompt section. In polyFORTH, this was accomplished by defining a new vocabulary and having it be the only one searched. in fig-PORTH, this option is not directly available since the FORTH vocabulary is searched after the current vocabulary. This may be solved by carefully breaking links with zero entries, or alternatively by defining a special dictionary search routine that stops at some fence value. I chose the latter.

It wasn't obvious until the implementation began that operators would require concatenation of their identifying strings. It was also decided that parenthesis would be placed around each level of expression nesting 80 that ambiguity could be eliminated without rearranging expressions for precedence. This occasionally leads to expression such as $((a r r a y+2)+2)$. This is unavoidable since even the constants within the expression are treated as strings rather than numbers. Thus, the example cannot be reduced to (array+4).

Error recovery is not nearly as good as I'd like it to be. Stack underflow in the diagraming session is generally fatal. Due to the amount of bookkeeping already being done, there is no provision for retracting answers after wrong data has been put on the diagram stack. This is inconvenient in a first pass through a function, but has not proved to be a problem once a feel for the tool and the function being diagrammed has been acquired.

Provision is left for user defined functions in the last two screens of the diagram source. This allows commonly used functions to be handled in an automated fashion. This makes it very easy to define composite functions such as, 1 - as the sum of its component parts. For outside of functions, constant and variable
have been redefined to put their own name on the stack. Before this facility was added, I always retyped the variable name manually when it came up.

The spool function and some of the source reading routines such DOC assume that screen blocks are contiguous 1,024 byte areas. Those functions using BLOCR will have to be rewritten if this is not the case in your system. I recomend that you instead generate a new system with $1 k$ buffers as that is faster and more flexible.

WEAKNESSES AND PROPOSED FUTURE EXTENSIONS

The diagrammer presently does not keep track of the contents of the return stack. This requires uses of $R>$ and $I g^{\circ}$ to the operator for clarification. Try a pencil for now. This could be added in a similar fashion as IF..ELSE..THEN by an additional stack.

The area of error recovery is ripe for suggestions. Perhaps some dummy buffer area could be added and tested in PSTAR. This would allow detection prior to destruction on stack underflow. Backing up by reading backwards would be nice but also very difficult to implement.

## CONCLUSION

Now that the tool has been built, its real function is more evident. It is still used for documenting words as originally intended, however, its primary usage is debugging and validating code. It has also proved to be very useful as a teaching aid to explain what is going on within the stack. I hope it will be as usefu: to you as it has been to me.

## A STACK DIAGRAM GLOSSARY

## VARIABLES

: BK The base block number for spooled stack diagram.
:LN Line number being printed. Used for page headings.
:SC Current screen number being spooled.

IFPTR The address of top of IF stack. Used to restore values on stack for IF...ELSE...THEN construct.

IFST The area reserved for pointers to previous stack contents. It is used to restore the stack on ELSE and THEN clauses.

SPL A temporary variable used by :NFD to retreat the spool file to erase the unknown stack prior to operator specification of what is added or dropped.

SPOOL Offset into spooled print file.
SUM The sum of differences in two strings. Used in -TEXT. Value is 0 for a text match and nonzero if different.

Tl Pointer to current input word in memory (type format).

## CONSTANTS

LLIMIT The limit address for dictionary search to keep from using standard FORTH words from within the STACK diagrammed words.

## FUNCTIONS

| $-\ldots$ | 'FIND pfa length true (found) |
| :--- | :--- | :--- |
| - | 'FIND false |

This is the same as -FIND except that the true condition is set only if the work is found above LLIMIT. This restricts the search to stack vocabulary words.


A string constant used for building expressions when arithmetic or logical operations are encountered in the diagrammed input string.

Defined back to its original state after being used as a concatenation token, this marks the beginning of a comment.

All text following it is ignored until the next ) .

Tests two strings for not equal. --- , ')'

A string constant used for building expressions when arithmetic or logical operations are encountered in the diagramed input string.

```
st1 st2 -TEXT cond
```

True if the two strings differ. val 1- val-1

Decrements the top of stack value by one.
v1 v2 2DROP ---
Drops the top 2 elements off the stack and discards them.
$\square$
This is the stack diagram redefinition of colon. It diagrams the word following it instead of compiling.

It is invoked by colon as the very last definition from within this package.
stl st2 : C st3

Concatenates two strings into a single combined string. It is used to build expressions when operators are encountered in the screen to be diagramed.
--- :HEAD ---
Prints the header for a line of output to the console and also the spool file.
--- :KILL ---

Removes and discards the top of the IFST.
--- :NFD addr
This is called when the word being analyzed is not in the special stack vocabulary. It checks for valid numbers. If this test is passed, it returns a pointer to that string. Otherwise, it invokes SKBD to get user help.
adr : PSH ---
Pushes the address of a level of the stack values onto the separate IF stack. This is used for IF..ELSE..THEN stack restoration and checking.
adr :RST ell el2 etc
Restores the stack from the IFST stack. Does not affect the IFST.
adr :SAV
Saves current stack element list on the IFST. Does not affect the parameter stack.

Marks the end of the spool file with a zero.
adr ?NUM cond
Checks current word to determine whether it qualifies as a legal hexadecimal number.

## --- CONSTANT ---

A defining word which causes the name of the defined word to be put on the stack when that word is encountered.
--- DEPTH depth
Computes the depth of the stack in items.
scr* DOC

Searches for a colon followed by the word whose name follows this invocation on the specified screen. It aborts if the definition is not on the specified screen. Otherwise, it commences to generate the diagram for the word specified.

## --- ELSE ell el2 etc

Clears the stack and then restores it from IFST.
-- ESC ---

Aborts the package if an escape key was the first key pressed in answer to the "PUSH?" prompt. The vocabulary reverts to FORTH; however, the stack diagram package is still loaded and ready to go.

## --- G-HERE adr cond

Moves a string from PAD into the dictionary. It allots the space and leaves the address of the item and a true cond if successful. It leaves only a false cond if no valid string was found.
expr G(1) op(expr)
Builds an expression from a simpler expression. At execution time of the following word, the top of the stack is enclosed in parenthesis and preceded by the operation symbol. It is used for unary operations. eg. -(name)
expr G(2) op(expr)
Similar to $G(1)$ except that unary operation is also enclosed within the parenthesis. eg. (name*2)
espr G(3) op(expr)
Similar to $G(1)$ except that binary operation is also enclosed within the parenthesis. eg. (vall+val2)
inadr
GBLD
An auxiliary word used to build a named string in the dictionary from the word following GBLD. This is used at
compile time of the stack diagram package. --- GWRD ---

A defining word for building strings into the dictionary at compile time of the stack package. On invocation of the new word, the address of the string displaying its own name is put on the stack. The word that follows GWRD is read twice at compile time, once for the name of the function, and a second time to be placed in string format into the dictionary. This is used to build up constant words for the diagraming package.
cond IF
Drops the condition flag from the top of the stack without evaluating it. It then invokes :SAV for ELSE restoration and THEN error checking.
adr cnt MTYP
Types the message to the screen and also passes the parameters to STYP for spooling.
src dst len MVB dst len srctl src
Intermediate function to set up for MVDEL.
sre dst delim MVDEL adr
Move a string from the source to the destination address until the specified delimiter is encountered. This is used to build data strings within the dictionary.
--- PDOC --
Prints the latest generated diagram from the default spool file blocks.
--- PHDG ---
Prints the top of page heading and sets the lines per page count. Used by PRTDOC.
blk\# PRTDOC ---

Prints the stack diagram from the spool file whose starting block is the specified blk*.
--- PSTAR
Prints all words from the string addresses on the stack. The top element is printed to the right of previous ¿冫ements. The stack is unchanged.
adr cnt PWRD

Prints one word via MTYP. Used by ?STAK.
--- REPEAT ---
Functionally identical to the redefined THEN.
ser\# SCRST scr*
Resets the spool pointer and places the screen number into the beginning of the output spool to be used in top of page headers by PDOC.
scri SDOC

Documents one whole screen by exezuting it, using the diagram definitions.
--- SKBD

This scans the keyboard for user interaction. It generates the "DROP?" and "PUSH VALUE?" prompts. It is invoked ohenever intervention is required in the ijagraming process.
=har SPIT
Writes character out to disk spool : 1e.

## --- STACK ---

This is the name of the vocabulary :ontaining this package.
adr cnt STYP

Similar to TYPE but spools to disk rather than typing to the screen. Outputs an additional two blanks after the message.
adr $T$; cond
Tests the current string for a match to the FORTH word semicolon. This is used to exit DOC.
-- THEN
Re-defined in the stack vocabulary, this cleans up the IFST. If the depth of the stack has changed from before the ELSE, it issues a warning and calls SKBD to allow the user to correct a stack depth disparrity between the IF and ELSE clauses.
--- VARIABLE
A defining word which causes the name of the defined word to be put on the stack when that word is encountered.
 or


```
SCM:110
    (atack diagras paokege oor it %. A. Cole a10326)
    -TEXI O Sum, SUAP, ce sum itrue if 2 atringa differ)
    DO OVEA 1.ce OVEM 1 - ce = SUM -I LOOF
    2080% Sum:
    MTEST OVEl DUP Ce it SWAP -TEKT ; ( zeal 2 words ror <>)
    DOC scast heaE 32 wond dup ce 1. allor (docweent (dar)
```




```
    HEILE a> 102%:
```




```
        1 now so rroe merel
    *e0* Prom: Barry A. Cole Los Angeles, CA 2t3-390-3851 ****
```

111 TRIAD


OK

## FORTH CLASS



Date: June 22-26
Where: Humbolt State University Arcata, CA 95521

Who: Kim Harris and Henry Laxen
What: Intensive 5-day course on the use of FORTH

Cost: $\quad \$ 100-\$ 140$ plus room and board
How: Call Prof. Ron Zammit (707) 826-3275

## MMS-FORTH FOR STRINGY FLOPPIES

Ralth Microsystems will make available to all licensed MMS-FORTH users a modified version that runs on the TRS-80 with an EXATRON stringy floppy. This modification is said to make MMS-FORTH operate as it would on a disk except for the speed. Users retain the capability to switch back to cassette operation with a single command. Implementation includes the normal read/write block commands plus a number of new utility words. The modification is available on ESF wafer for $\$ 14.95$ including shipping. For more information contact:

[^0]How to form a FIG Chapter:

1. You decide on a time and place for the first meeting in your area. (Allow about 8 weeks for steps 2 and 3.)
2. Send to FIG in San Carlos, CA a meeting announcement on one side of $8-1 / 2 \times 11$ paper (one copy is enough). Also send list of ZIP numbers that you want mailed to (use first three digits if it works for you).
3. FIG will print, address and mail to members with the ZIP's you want from San Carlos, CA.
4. When you've had your first meeting with 5 or more attendees then FIG will provide you with names in your area. You have to tell us when you have 5 or more.

## Northern California

| 4th Sat | FIG Monthly Meeting, $1: 00$ |
| :--- | :--- | :--- |
|  | p.m., at Southland Shopping |
|  | Ctr., Hayward, CA. FORML |
|  | Workshop at $10: 00$ a.m. |

Southern California
Los Angeles
4th Sat FIG Meeting, 11:00 a.m., Allstate Savings, 8800 So. Sepulveda, L.A. Call Phillip Wasson, (213) 649-1428.

Orange County
3rd Sat FIG Meeting, 12:00 noon, Fullerton Savings, 18020 Brockhorst, Fountain Valley, CA. (714) 896-2016.

## San Diego

Thur

FIG Meeting, 12:00 noon. Call Guy Kelly at (714) 268-3100, $\times 4784$ for site.

Massachusetts
3rd Wed MMSFORTH Users Group, 7:00 p.m., Cochituate, MA. Call Dick Miller at (617) 653-6136 for site.

Seattle Chuck Pliske or Dwight Vandenburg at (206) 542-7611.

Potomac Paul van der Eijk at (703) 354-7443 or Joel Shprentz at (703) 437-9218.

Tulsa Art Gorski at (918) 743-0113.
Texas Jeff Lewis at (713) 719-3320 or John Earls at (214) 6612928 or Dwayne Gustaus at (817) 387-6976. John Hastings (512) 835-1918.

Phoenix Peter Bates at (602) 996-8398.

New York Tom Jung at (212) 746-4062.
Detroit Dean Vieau at (313) 493-5105.
England FORTH Interest Group, c/o 38, Worsley Road, Frimley, Camberley, Surrey, GUl6 5AU, England

Japan Mr. Okada, Presdient, ASR Corp. Int'1, 3-15-8, NishiShimbashi Manato-ku, Tokyo, Japan.

Canada Quebec

Gilles Paillard at (418) 8711960.

West Germany Wolf Gervert, Roter Hahn 29, D-2 Hamburg 72, West Germany, (040) 644-3985.

## Publishers Note:

Please send notes (and reports) about your meetings.


## FORTH DIITIETSIIOIS



FORTH INTEREST GROUP


54
53 Increasing Disk Access Speed


35
Letters

40
Technotes

45
Data Base Design


60
FORTH Vendors

61 Decompiler

Chapters/Meetings

Published by Forth Interest Group

Volume III No. 2
Publisher
Editor
Editorial Review Board
Bill Ragsdale
Dave Boulton
Kim Harris
John James
Dave Kilbridge
Henry Laxen
George Maverick
Bob Smith
John Bumgarner
Bob Berkey

July/August 1981
Roy C. Martens
C. J. Street

FORTH DIMENSIONS solicits editorial material, comments and letters. No responsibility is assumed for accuracy of material submitted. ALL MATERIAL PUBLISHED BY THE FORTH INTEREST GROUP IS IN THE PUBLIC DOMAIN. Information in FORTH DIMENSIONS may be reproduced with credit given to the author and the Forth Interest Groud.

Subscription to FORTH DIMENSIONS is tree with membership in the Forth Interest Group at $\$ 15.00$ per year ( $\$ 27.00$ foreign air). For membership. change of address and/or to submit material, the address is:

Forth Interest Group
P.O. Box 1105

San Carlos, CA 94070

## 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 $\mathbf{2 , 4 0 0}$ worldwide. It was formed in 1978 by FORTH programmers to encourage use of the language by the interchange of ideas through seminars and publications.

## EDITOR'S COLUMN

The feedback on our new applications editorial policy is all positive. To date, we are receiving a nice variety of articles. I want to urge our members not to slack off. In order to keep up a steady flow of quality output, we need quality input - IN QUANTITY.

If you have an article you have been meaning to write, please get it down and send it in. If you havr an application, programming trick or tool that you have found useful, please share it with our members. Remember: YOU DON'T HAVE TO BE A WRITER-our staff is set up to help you with whatever you need to make your idea publishable.

Please send all submissions to:
Editor
FORTH DIMENSIONS
P.O. Box 1105
San Carlos, CA 94070
HEX is back this month, and there are photos of the Rochester Conference courtesy of George W. Shaw, II. We are always looking for photos (black and white or color prints preferred) and cartoon ideas, too.

Starting next edition, FORTH DIMENSIONS will have a marketing column in a question and answer format. It you have had ideas, programs, etc., that you wondered how to sell, this column will be for you. Please direct your marketing questions to the above address. Questions of general interest will be answered in this column by experts chosen for their knowledge of marketing and computer hardware and software.

## C. J. Street <br> Editor

## PUBLISHER'S COLUMN

Lots of good news! The reaction to the application orientation of FORTH DIMENSIONS has been very positive. Thanks to our editor, Carl Street. The more articles you send Carl, the closer we come to being able to go monthly. Our plans are to make FORTH DIMENSIONS more general interest and publish high level (sic) technical material twice a year, ala, 1980 FORML Proceedings.

Plans for the 1981 FORML (FORTH Modification Laboratory) Conference are underway. Refer to page 63 for more details. The FIG National Convention will be on Saturday, November 28th in the San Francisco Bay Area. Make your plans.

Now, some bad news! We have to raise some of our prices. It's been a couple of years since we've done any price adjusting and cost increases have caught up with us. The order form on the last page reflects the new costs which are now in effect Sorry, we'll do our best to hold the line.

## Roy Martens

## LETTERS

Jear FIG,
Please find enclosed two short articles mich might be suitable for publication in FORTH DIMENSIONS.

I did not ask for the publication kit, s) I hope the articles do not violate your =sles too much. Second, my native language is not English but Dutch, so forgive se if there are any errors and feel free $:=$ correct them.

Please note my new telephone number and zorrect it in your listing of local FIG :hapters.

We have not had many meetings lately, probably because our members are too Eこtive!

Paul van der Eijk
5480 Wisconsin Avenue 1128
Chevy Chase, MD 20015
(301) 656-2772

Thank you for your articles, Paul. jeaders can find them under the appli=ations area of this issue.--ed.

Sear FIG,
I recently purchased a listing of fig?ORTH for the 8080 from you and I am very mpressed with the Language package. You $\because 11$ find enclosed an order and Bank Draft Eor several books which I eagerly await. : received my Dual Micropolis Mod II Disk Eives only two weeks ago and my first zeoject was to assemble FORTH. The disk :aterface routines were quite easy to link :s the Micropolis DOS using ideas from the TPM interface supplied. However, when I :ried to LOAD a short word definition off I screen the system would lock up and not :ome back with any error messages or the ' $\partial \mathrm{R}^{\prime}$; because the system would compile *ords from the keyboard and the Disk I/O wrked well, I was puzzled as to why there -is a problem. After four days of search-
ing and debugging, $I$ found that the program was looping through INTERPRET, and each time the parameter stack had an extra value on it. Eventually, I found the bug; it was in the ENCLOSE routine and the problem is that only an 8 bit counter is used to hold the offset into the buffer. However, the Micropolis sectors are 256 bytes and so are my Forth Disc Buffers. If there are any non-delimiter characters in a buffer, then all works OK. However, if the buffer holds 256 blanks, then the loop around ENCLI scans to the end of the buffer but the 8 Bit offset ends up pointing at the start of the buffer still an INTERPRET never gets to to see the NULL at the end of the buffer. Obviously, the routine works OK for CPM 128 Byte Sectors, but needs modifying for larger capacity sectors.

I have included the source listing for ENCLOSE as modified by me (sorry, I haven't got my printer going yet). I have used the DE register pair for the offset counter and have kept the definition character in the Accumulator which means pushing and popping it when it is necessary to check for a NULL.

I hope you find this of interest and maybe you will include a change of this sort in future versions. I learned a great deal from this problem, and it was probably to my advantage that it occurred, as my only prior information was the 'FORTH' BYTE. I really learned the hard way.

William D. Miles
P. O. Box 225

Red Cliffs
Victoria, 3496
Australia
Thank you for your contribution. NOTE: You will find Mr. Miles' bug fix in the TECHNOTES, BUGS \& FIXES section of this issue.--ed.

DON'T MISS OUT! GET YOUR PAPER IN
EARLY FOR THE FORML CONFERENCE!

Dear FIG,
Could you print my address in your next FORTH DIMENSIONS issue: I would like to hear from other Belgian FORTH-ists!

```
Michel Dessaintes
Rue de Zualart 64
B 5810 Suarlee Begium
```

OK, Michel, start watching your mail box!--ed.

Dear FIG,
Congratulations on your last issue (Vol. II, No. 6). It's nice to see some tutorial inputs at a level that beginners like me can understand. Reep it up!

Would you please print the SEARCH routine mentioned in John James' article on page 165 of Vol. II, No. 6. It apparently got replaced by the correction notice at the bottom of the page.

I was interested in trying EDGAR $H$. FEY'S FEDIT in Vol. II, No. 5, but was stumped by the word REPL which was not defined. Is it possible MR. FEY could provide the definition? (Also, I noted that SCR $\$ 67$ errors at line 48 -B/BUD which apparently is supposed to be B/BUF.) Screens should be required to be loadable, not edited by publisher or author without loading edited version.

In respect to editing, please also note that Major Selzer's article in the Vol. II, NO. 3 issue on page $83, \operatorname{SCR}+200$ line 8 should apparently be 08 CASE for left cursol as opposed to $O B$ as printed, since $O B$ is used for UP cursor. This screen does work when above mentioned change is made.

I realize that submitted copy may need to be retyped but the dangers of intro-
ducing errors are ever present. I'm sure that you catch most of them.

Robert I. Demrow
P. O. Box 158 BLUE STA.

Andover, MA 01810
Thank you for your thoughts. Glad you like our new approach. John James SEARCH is in a previous issue. Regarding errors, we do try to minimize them; but we are only human.--ed.

Dear FIG,
During September of 1980 , material was ordered which included hard copy of figFORTH for the Motorola 6802 (6809 preferred) CPU, and FIG membership for a year. Hard copy received was Talbot Microsystems v.1.1. 6809 FORTH. After a considerable amount of study, and a complete rewrite, that software is now up and running, apparently as designed. (No operating bugs have been detected, but it would be reasonable to expect bugs to appear far into the future.) Some general comments on the system may be of interest.

A major factor in the acquistion of this software was the indicated ability to run high level software on a small system. If the Talbot software is designed for a microsystem, then $I$ must have $a$ nano-system by definition; a disk would cost far more than all hardware currently in use, and appears quite unrealistic for this home hobby system. The alternative cassette is implemented, but patience would be strained beyond limits if nearly 8 R words were loaded for each use at 300 baud. Thus, my system clearly demands use of EPROMs for source code.

I have used several methods of code reduction.

1. A short branch to several copies of NEXT.
2. Place the user area in the direct pad.
3. Add a byte literal as well as LIT.
4. Some high level routines are shorter in code.

The end result of this process was code retaining nearly all of Mr. Talbot's word definitions, and fitting easily in 6K bytes ( 3 2716's). There is very little jenchmark information available (this would make a worthwhile FORTH DIMENSIONS article), but those found generally ran in $1 / 2$ the time cited for the APPLE.
A. R. Gunion

182 Minuteman Drive
Concord, MA
The real definitions of nano and micro as applied to systems vary with each ser. Suffice to say that FORTH is by jefinition a disk based system. If you do not have a disk then you are compromising In an area vital to obtaining the real potential of the system.

Regarding benchmarking, it has always seen the position of FORTH DIMENSIONS that the nature of FORTH makes benchmarks more 2f an indication of the speed of a CPU :ian any particular system and we genera!ly do not publish them. This has been siscussed at length in previous edi-:ions.--ed.

Jear FIG,
While I cannot disagree with the intent of "An Open Response" in FORTH DIMENSIONS, $\because o l$. II, No. 6, concerning the hardware -equirements for FORTH, I feel you may siscourage some with the categorical statements you made. It is possible to accomplish a great deal with much less than you described. I hand-installed the 5502-verison of fig-FORTH on a homebrew, KIM-based system that had only 8 K of RAM and traditional cassette-storage. My "terminal" was a memory-mapped 16-1ine by 32-character display with ASCII keyjoard. This minimal system has given me hours of pleasure and practical experience dith FORTH, and because of the concise nature of FORTH has been capable of power-
ful constructs. An acquaintance has installed a cut-down version on a 5 K KIM with ASCII keyboard and walking "timessquare" display on the KIM LED's. There is no question that we would be more comfortable in the hardware environment you define, but compared to Tiny-Basic, for example, these minimal FORTH's are heaven.

I found the same bugs in the May 1980 6502-version of fig-PORTH that Grotke and McCarthy have already reported. In addition, I would warn prospective installers that the TRACE routine depends on the output routines preserving the $Y$-register, and that the MON routine is not quite correct. Since the 6502-processor increments the program counter by two when BRK instruction is executed, BRR should be followed by a NOP to ensure that a simple machine-language monitor will return to the start of the IDX XSAVE instruction.

My system now includes a $320 \times 200$ dot raster-scan display, and $I$ am interested in corresponding with others concerning FORTH-based graphics processors.

Kent A. Reed<br>49 Midline Court<br>Gaithersburg, MD 20760

The point of the "Open Response" was not to condem anyone's system; rather to point out that FORTH is designed to be used with a disk. Naturally, the nature of FORTH means that it will perform (and outperform other languages) regardless of the environment. Your "bug" comments are appreciated.--ed.

Dear FIG,
In bringing up the 6502 Assembly Source listing on my Rockwell System 65, I encountered a problem involving writing or reading the disk drives. The symptoms involved setting an 01 error everytime the disk was asked to jump to the next track.

The problem turns out to be hardware and only exists on a Sys 65 with Pertec
model FD200 drives. The fix is simple and is detailed in Rockwell Service Bulletin 'SYSTEM 65-7' which may be obtained by writing:

## Rodger Doerr <br> SYSTEM 65 Customer Service Dept. ROCKWELL INTERNATIONAL

Microelectronic Devices
P. O. Box 3669

Anaheim, CA 92803
(Or call Rodger at (714) 632-2862.)
I hope that this information can be helpful to other individuals who are working with FORTH on the SYS 65.

> Jack Haller 230 Mechanic St. Boonton, NJ 07005

Thank you--I am sure you have saved more than one frustrated programmer a few sleepless nights.--ed.

## Dear FIG,

Enclosed is $\$ 12.00$ (now $\$ 15.00-$ Pub.) for another year of FORTH DLMENSIONS. I have FORTH up on 2 KIM's (Dean's version) and a Superbrain; although my "playtime" is limited, I enjoy tinkering very much. It might amuse you and Mr. Moore to know that one of the systems is going to control a $10^{\prime}$ dish radiotelescope which $I$ also use for looking at thunderstorms.

I an slowly getting together parts of a Western Digital-based computer. Their $p^{-}$ code chip is a natural for FORTH--almost all primitives are single instructions. This is a very long-term project and, no doubt, someone will beat me to it, but it needs doing. Please pass this on to anyone who might be interested. I would be glad to correspond with them.

As a long-time but not prolific user of FORTH, I'd like to put in my buck's (inflated two bits') worth: KISS--this acronym is keep it simple, stupid. In other words, let's not get too many words
into "Basic FORTH" vocabulary. Certainly, more advanced words are useful and should be published and documented, and are, of course, part of the FORTH vocabulary by definition. Any standards, however, should be kept very simple. Enough.

Don Latham Six Mile Road Huson, MT 59846

OR interested members, drop him a line.--ed.

Dear FIG,
This letter is in response to $C$. A. McCarthy's letter in FORTH DIMENSIONS, Vol. II No. 6 concerning the errors he listed:

## Page 0061

Yes, there should be a SEMIS at the end of the UPDATE.

Page 0064
I haven't hooked up disks to FORTH yet, so I didn't notice this one, but $I$ agree that the displacement in line 3075 is wrong.

Page 0067
I dropped one of the STX XSAVE's without ill effect.

Page 0069
The extra SEMIS is superfluous, but will not have any harmful effect.

I did find another error in the listing. This one, rather than being a typo, appears to be an error in program logic.

Page 0017, lines 0803-0805. The listing for routine ZERO shows:

LDA $0, X$
ORA 1, X
STY $1, X$
Since $Y$ contains 0 at this point, the zero flag in the processor status register will always be set by the STY instruction. Therefore, the branch which follows will never be taken, resulting in a logical "false" value always being left on top of the stack. I replaced the above code with the following:

LDA $1, X$
STY 1, X
ORA $0, X$
This causes the processor status to be set properly to indicate whether the top stack entry is a zero or not. I know of no other errors in the listing.

> Steve Wheeler
> 504 Elmira
> Aurora, CO 80010

Thank you for passing along the above.--ed.

Sear FIG,
A little note about changes in the situation in NW Europe. During the second =alf of March, there was an exhibition in Nalmo (close to Copenhagen) - "Datacraft : : ' (Computer power-81).

Up until then, FORTH was very difficult :s get in touch with here in Sweden. To IF great astonishment, there were at least -. perhaps 6, systems running in different =ools. The most interesting one was a Foly-FORTH system running on an ABC-80 (a沙 z-80 lowend machine). There were also E:g-FORTH's running realtime setups on ?ET's.

To me, who had up until then been 'dryswimming ' FORTH, it was quite an experi-
ence to key in definitions, clear, compact, and (CR/LF), to be able to use them. Quite a kick!

## Calle Hogard

Glad to hear things are moving ahead.-ed.

Dear FIG,
Response to "An Open Response".
I object strongly to the tone of the above (unsigned) article in Vol. II, No. 6. It is the attitude of the 'computer professional' with access to a large, all singing, all dancing computer looking down his/her nose at the pathetic squirmings of the home computer buff. If this attitude had prevailed, there would be no cheap computers. As it is, a lot of harm is still done by designers making their small computer systems in the image of large computer systems instead of making them like super calculators.

Like many others, I first became interested in FORTH via the August ' 80 issue of BYTE. One thing that attracted me was the idea that here was a high level language which could be used over the whole range of hardware. There are obvious resemblances between the FORTH and the $H P$ programmable calculator languages and it is reported that FORTH or similar languages are used in hand-held language translators and in one of the hand-held computers. Compare the editorial and, more specifically, Charles Moore's "Characteristics of a FORTH Computer" ( p .88 ) in that BYTE issue with your "Open Response". FORTH is a language in which the user is allowed unparalleled freedom. Please do not insult us by drawing arbitrary limits which will in any case be out of date in a short time.

I will agree that a quart cannot usually be fitted in a pint pot. Solution: devise a means of listing the glossary in such a way that for any word, the indirectly referenced words underlying it
can be read. The answer to those wishing to devise minimum systems would then be "go away and get on with it!" Remember that necessity is the mother of invention and the professionals are those who carry on in the wake of the amateurs-- like Einstein--to name but one.

> N.E.H. Feilden
> 47 London Road
> Halesworth Sufolk IP19 8LR England
P.S. Number typing (e.g., Fixed, floating, double, quad precision, etc.) Surely, all this business of having hundreds of different numbers types is silly, cumbersome, and FORTRAN-like.

Why not forget the whole scheme and do it like BASE. That is to say, have a constant, say NTYPE which tells all operators how many bytes to operate on and whether fixed or floating. It would, of course, be necessary to code all constants and variables in the same way so that when referenced, the appropriate conversions would be done. If this were done in linked lists, then the memory overhead would be very small. The whole thing would be vastly easier to use than what is currently proposed. This suggestion would help to reduce the number of words to remember.

Sounds like you have some interesting and creative approaches to problem solving. You might be interested to know that the author of "Open Response" works on a home size computer. I am sure that no offense was meant and if the author of "Oper Response" would like to answer in this space or another colum, we will be glad to print it.--ed.

## HELP WANTED

Los Angeles Area FORTH PROGRAMMER WANTED -- Contact Linda Stoffer at Pace Personnel, (213) 788-7039.

PORTH, Inc. has the following job openings:

# TECHNOTES, BUGS, FIXES 

TIPS ON BRINGING UP 8080 Fig-FORTH

Ted Shapin<br>5110 E. Elsinore Aenue<br>Orange, CA 92669

Some of the "gotchas" I ran into in bringing up 8080 Fig-FORTH may be helpful to others.

Make sure your assembler will handle lines such as DW A,B-\$ correctly. The Boston Systems Office cross-assemblers use the address of the first operand as the value for "\$" in the second operand. This leads to a system that will print out the sign-on message but will fail to perform many other operations correctly. I got around this by changing such occurrences to two separate lines: DW A and DW B-S.

The next problem to solve is how to type in the editor screens. It is nearly impossible to type the editor in twice correctly. As R. Allyn Saroyan pointed out, you only need to type in a minieditor twice. Once, to get it in the dictionary so you can use it, and again, to get it to a screen so you can put in on disk. The mini-editor is simply taken from the implementation model editor screens as follows:

HEX : TEXT HERE C/L $1+$ BLANKS WORD HERE PAD C/L 1+ CMOVE ;
: LINE DUP FFFG AND 17 TERROR SCR @ (LINE) DROP ;
: -MOVE LINE C/L CHOVE UPDATE ;
: P 1 TEXT PAD $1+$ SWAP -MOVE ;

## DECIMAL

Now, proceed to use it to write itself to the disk. You can do this by picking an unused screen, say 85 and typing 85 LIST. Now use "P" to place a line of text on the screen, e.g., 0 P (Mini-editor)
W. 11 place a comment on line 0 of the :urrent screen.

Type the rest of the lines above and :hen use the word "PLUSH" to write the mini-editor to disk. Now, when you need $=0$ start the system again, just type 85 OAD and your mini-editor will be put into :ie dictionary.

Use the mini-editor to type in the FigPORTR editor. The string search screen =an be omitted if you do not have a version written in highlevel PORTH.

## USING ENCLOSE ON 8080

Using ENCLOSE with disk block buffers of 256 bytes each or larger on the 8080 processor.


NOTE: see Mr. Miles' letter in Letters section.--ed.

Mr. William D. Miles
P. O. Box 225

Red Cliffs
Victoria 3496
Australia

## CORRECTIONS TO METAFORTH

> John J. Cassady 339 15th Street Oakland, CA 94612

The following corrections to the FigFORTH cross-compiler, METAFORTH, by John Cassady should be noted:

page 26 screen 66 line 7 should read<br>KISR H LXI SRA5 SHLD 12 ORG + LBLD SPHL NEXT JMP

page 38 dumped memory location 798C should be 6A

A few lucky purchasers will have noted that they possess those rare copies of MRTAFORTH in which pages 8 and 9 are swapped.

METAFORTH, by the way, is a crosscompiler for Fig-FORTH. It can be used to regenerate a FORTH system including the nucleus without resort to an external conventional assembler. This is helpful when modifying low level words, generating "stand-alone" applications, converting to FORTH-79 and the like. A special section is devoted to generating headless configurations with the same or different processor.

METAFORTH is available in hardcopy through: MOUNTAIN VIEW PRESS, PO Box 4656, Mountain View, CA 94040 for $\$ 30.00$. There are plans to have it available on disk and compatible with several of the popular commercial figFORTHs from their respective vendors.

# CHANGING 8080 fig-FORTH FOR DISK COPYING 

Ted Shapin<br>5110 E. Elsinore Avenue<br>Orange, CA 92669

The FigFORTH 8080 implementation uses all bytes of all sectors on the single and double density diskette. This means 2002 sectors on a disk for single sector and 4004 sectors for double. This is not a multiple of eight so the last screen on a disk will be split across two disks. By simply changing the equates for SPDRVI and SPDRV2 to 2000 and 4000, we will have an even number of screens per disk. This allows a screen disk to be copied from disk A to disk $B$ by using the Fig-FORTH COPY word.

NOTE: Ted has the correct method. Any other system setup that could split screens is incorrect.--ed.

## FORTH STANDARDS CORNER

Robert L. Smith

There is a need for a channel of communication regarding the standardization of FORTH. A major topic is the clarification of the FORTH-79 Standard. What changes are required or desirable for clarification or extensions to the Standard? Is the FORTH Standards Team the appropriate mechanism for obtaining a "seal of approval" for corrections and changes to the Standards?

Let us first consider a fairly simple topic, the unsigned count specified in the definition of FILL in the 79-Standard. FILL is defined as follows:

FILL addr n byte 234

Fill memory beginning at address with a sequence of $n$ copies of byte. If the quantity $n$ is less than or equal to zero, take no action.

This is a clear and reasonable unambiguous definition. However, at the Rochester FORTH Standards Conference, there was a strong consensus that the byte count $n$ should be an unsigned number. The restriction in the definition seems to be unnecessary; the only thing to be said in its favor is that it might save a programmer from an inadvertent error (and generally FORTH does not try to save programmers from their errors). If the unsigned FILL were to be the fundamental definition, then the signed version would be trivial to implement. The reverse is more difficult. Thus, the unsigned FILL would lead to better "factoring". Furthermore, a common use for FILL is to preset a large portion of memory. The unsigned version is clearly better suited for this task.

Having said that, what should be done? Since the current definition is unambiguous, and since 79-Standard versions of FORTH currently exist (with several more in advanced stages of development), it seems to me that there should be no change to the 79-Standard in this area. The Standard Team has suggested one mechanism for evolutionary changes in FORTH via "Experimental Proposals". An experimental program would, however, involve a new name for the changed function and could not become a permanently accepted change until two revisions of the Standard. That may or may not be acceptable, depending on the frequency of the revisions.

Please send in material, questions, and comments relevant to FORTH Standards. I will try to cover one or two areas with each issue. Possible topics for next time are the words WORD and +LOOP.

## CORRECTION

"Systems Guide to fig-FORTH" by Ting is not available through FIG. Orders for this book, revised lst edition @ $\$ 25.00$, should be sent to:

MOUNTAIN VIEW PRESS
PO Box 4656
Mountain View, CA 94040

## NEW PRODUCTS

SYM-1 FORTH
Saturn Software Limited has implemented Fig-FORTH for the SYM-1 single board computer. Their implementation takes advantage of many of the features and resources of the SYM-1.

SYM-FORTH 1.0 (disk version) requires 16 K of ram, serial terminal, and the dual HDE mini disk system. System has been upgraded to the 79-STANDARD and includes a versatile input line editor, fig-style editor, 6502 assembler, and a cassette interface. This product is also supported by a quarterly newsletter with an initial circulation of 100 .

Extras included:
Assembler, editor, cassette interface, plus numerous utilities and demos presented through subscription to newsletter.

Machine on which product runs:

SYM-1, 6502 singleboard computer.
Memory requirements: 16 R of ram

## Manual:

The 74 page manual includes introductory tutorial material, system information, and glossaries for the FORTH, EDITOR, and ASSEMBLER vocabularies. The manual is available separately for $\$ 25$ which will be credited towards a later purchase.

Form product is shipped in:
Product is distributed on two 5-1/4 inch diskettes, and boots with 79-STANDARD upgrade installed. (Cassette version is also available which can be upgraded to a disk system at any time.)

Product has five active installations of the disk version (79-STANDARD). There are also 50 installations of the cassette version.

## Price:

SFD-1 SM FORTH FOR DUAL HDE MINI DISK SYSTEM $\$ 150$ U.S., includes shipping, tax, etc.

## Vendor support:

Direct personal support by phone, correspondence, and newsletter.

Order turn around time:
Immediate.
For more information, contact:

Jack W. Brown<br>SATURN SOFTWARE LIMITED<br>8246 116A Street<br>Delta, B.C., V4C 5Y9, CANADA<br>(604) 596-9764

OSI-FORTH 2.0 / FIG-FORTH 1.1
This is a full implementation of the PORTH Interest Group Version 1.1 of FORTH. It runs under OS-65D3.12 (or 3.0, 3.1), on any disk-based Ohio Scientific system, and has access to all DOS commands and resources.

Extras include resident text editor, Assembler, and utility screens for transferring the system to a new disk, initializing library and system disk block storage tracks, copying screens from disk to disk, and reconfiguring the system memory usage.

## Machines:

Ohio Scientific C4P MP, C8P DF, C3, C28P DF, C1P MF, and C4P DF. While only one drive is needed, dual drives are supported.

Memory Required: 24K
Manual:
Currently 95+ pages--with new OSI-FORTH Letters added as they are produced. Twenty-four pages of discussion of particulars for OSI, utility screens, and operation of the editor (includes sample edit screen). FIG Installation manual included. Listings of utility and other sample screens. Available separately for \$9.95, which is credited toward system purchase.

Media Available: Eight-inch or mini disk.
Approximate number shipped: 25
Price:
\$79.95 includes shipping. (Florida residents add $4 \%$ sales tax.)

Delivery: 30 days.
Support:
OSI-FORTH Letters subscription available for $\$ 4$ per year. Contains fixes for any new minor bugs that may be found, as well as listings of application screens donated by users, or developed by Technical Products.

For more information, contact:
Daniel B. Caton
TECHNICAL PRODUCTS COMPANY
4151 N.W. 43 St., \#507
P. O. Box 12983

Gainesville, FL 32604
(904) 372-8439

## NEW PRODUCT

## DATRICON FORTH

Datricon now offers D-FORTH, a software package designed for use in conjunction with microprocessor-based, STD Bus com patible products using a Single Board System concept and offering a variety of 68xx/65xx processors. Datricon's single board controllers use interface standards such as the STD Bus, RS232, and RS422 for serial communcations and with or without parallel $I / O$ compatible with the popular isolated AC/DC module racks.

For more information, contact:
DATRICON CORPORATION
7911 N.E. 33rd Drive
Portland, OR 97211
(503) 284-8277

Warning--this FORTH is different in names and omitted 'vestigal words'.--ed.


## ELEMENTS OF A FORTH DATA BASE DESIGN

by Glen B. Haydon

In this day and age, data base design and manipulation is one of the major activities best accomplished with computers. In practice, FORTH proves to be an ideal language for developing and using custom data bases. By comparison with other languages, high or low level, FORTH is a winner. It meets the requirements of being interactive and providing documentation as identified by Fred Brooks in his book, THE MYTHICAL MAN-MONTH, as being ideal for the development of new systems. The amazing speed and ease with which custom data bases can be developed, more than justifies the effort required to learn FORTH.

I have developed a number of small data bases of up to 800 records containing 128 bytes each to serve my specific needs. I have also initialized, with simple formatted input and output routines, a custom data base for inventory control in a few hours one evening. Having used languages other than FORTH for similar work, it is highly unlikely $I$ will ever go back to them.

This discussion presents a group of utility FORTH word definitions which facilitate the development of custom data bases and a sample application using these utilities to define a small file. A number of techniques available in FORTH are illustrated.

Some months ago, at a regular monthly meeting of the FORTH INTEREST GROUP in Hayward, the prime mover of the group distributed and discussed several FORTH Screens which provided the foundation for beginning the definition of a data base file. I have modified his Screens slightly and expanded them to provide a general framework with which to define custom accounting data bases. I will assume that the reader has some knowledge of the fig-FORTH Model and proceed with the examination of Screens developed from it. In the discussion, FORTH words are
enclosed in single quotes to set them apart from the English words in the text. In FORTH, these words are used without the single quotes.

```
SCR 21
    * (SCRERN 21: cOMmENTS AND LOND FOR DJNO DATA FILE)
    127 EMIT 69 EMIT CR CR CR CR CR
    2." This deconstration daca system provides a pattern for the"
    2 *Tis demonstration daca system provides a pattern for the"
    CR. Eurther dovelcpment of any type of data base. The basic"
    CR ". Elle formating definltions are on Scremss 22 and 23. Some
```



```
    7CR ." utilities are on 2%. This modul should get you gtarted."
    8 : mROCEDE CR CR " ENTER 'Y' TO LOND SCRENS " KEY E9 = IF
    2922 DO I LOND coOp ENOIF
    |rocede
    1/ ;
SCR 22
        - scmem 22: FILE DEVECOMPT
        Varuble RECf, (holds the current record maber)
        * Variable orem (pinta to current file dencriptor)
        24 ONP 2+ E SWAP e: (dowle fetch )
        Cayour ( leeve bytea/racord-2, and bytes/block-1)
        OPEN & 4 + 2e
        REND ( m-th record, on stack, is mede current)
        - max oup orta e 2+ < < -
        IF - FILE ERGOR " OUTT TITN RDE! !:
        RECONO ( lewve addreas of n-th record)
```



```
        nOCRESS ( leave eddreas of the curtent record)
        rEci e mecomo.
    15 ;S
```

The first two Screens provide eight utility FORTH words for developing a data base file. The comments included in the Screens within parentheses should, combined with the memonic nature of the words, give you a clue to what is happening. The first two words are variables used in manipulating the file, 'RECF' and 'OPEN'. '2@' is a FORTH word, and alias for 'D@', which fetches the next two values beginning with the address on the top of the stack and places them on the stack. The word, 'LAYOUT', places two parameters of the new definition of a file on the stack for subsequent use. 'READ' is the first word that one will have occasion to use in routine manipulating records in the data base. It takes the number of the desired record from the top of the stack and, after checking to see that it is a valid record, places its value in the variable 'REC\#' which is used to identify the record then under consideration. The word 'RECORD' takes the value for a record number from the stack and returns its address to the stack.

Finally, 'ADDRESS' takes the record number at the variable 'RECf' and using 'RECORD' leaves the address of that record on the top of the stack. With only these eight FORTH words: two variables, one utility word, and four basic words for file referencing, we can proceed to the definition of three defining words in the next Screen.


The three words on the next Screen are called defining words because they are used to define new FORTH words as the names of fields in our record and to define the name for the file we are defining, each with specific properties. These words utilize the combination of the FORTH primitives '<BUILDS' and 'DOES>' which are present in the Model. It may take some time to fully appreciate what these primitive words accomplish and the way they work. Perhaps an examination of what they are doing in this Screen will help you understand their function.

Two types of record fields are distinguished and defined with separate words, a numerical or data field and a text field. The first word, "DFIELD', is used to add to a record being defined, a field containing the number of bytes given on the top of the stack and gives that field a name. In subsequent use, that newly defined word (data field name) will cause the address of that field in the record whose value is curreatly in the variable 'REC\#' to be left on the stack. This word is used to identify the location in a record where a numerical value is to be stored in a binary form. I call it a "data field", in contrast with a "text field" in which the length of the field should also be immediately available. Thus 'TFIELD' is used to define a "text
field" which will identify a field in the new record with a length in bytes given on the top of the stack and gives that field a name. In subsequent use, that newly defined word (text field name) will cause not only the address of that text field in the record whose value is currently in the variable 'RECf' to be left on the stack, but also the length of that ficld. The length is convenient when the primitive word 'TYPE' is used to print the character string in that field. Obviously the length is not needed in a data field, Thus, provisions are made for defining two types of fields in a record. As new fields are added to a record in the course of its definition, the current length of the record is maintained on the top of the stack.


Once the definition of the fields in a record is completed, the value of the record length remains on the stack. To this we need to add values for the number of records we wish to include and finally, the block number in which the records are to start, before we can use the defining word 'FILE' to give the file a name. Later when the new file name is used, the address of the necessary file parameters is placed in the recently defined variable 'OPEN' as required for access to any given record with the words defined in the first Screen.

With these two Screens, we have the file utilities necessary to define a new file. However, several characteristics of the particular implementation of FORTH which is being used are important. Most systems created under the Model have 128 bytes per block although any multiple of 128 can be used. In these sytems then, the largest record length can be no longer
than 128 bytes, but with a larger block size, larger records can be used. In order to take maximum advantage of the block size, it should be very nearly equal to a multiple of the record length. For example, a record length of 70 bytes would not leave enough room in a 128 byte block for a second record and in this case, 58 bytes of space would be wasted. If need be, such a designed file would work, but at the expense of memory space. Also, the initial block to be used in the definition created by the word 'FILE' must be chosen according to the block size for the particular implementation. For example, block 400 in an implementation with 128 bytes per block would be block 50 in an implementation with 1024 bytes per block. Although, I find a block size of 1024 to be more efficient and use it routinely, the Screens presented here have been written for and tested on an implementation with 128 bytes per block.

Before starting with a discussion of an example of the application of these file development utilities, several Screens of utilities for use in the input and output of numerical data will prove to be most helpful. These include a group of double precision utilities, date compression and expansion routines, a numerical routines for handing dollar amounts and storing them as double precision integers.

```
ScR 025
    CR - SGRED 25: DATE CONRRESSION AND EXPAMSION
        Datgit (comverts date inpue to 2 bytes)
```



```
        OERY 47 WORD hERE N(MEER 47 WORD HERE mmBER BL wOND
        HIRE NMAER DNTEEIT
```



```
            (\) | 47 HOLD # 47 HOD ; %) TYPE:
    :s
```

The double precision integer utilities are used in date compression and expansion as well as in the double precision integer operations for dollar amounts. These are simple extensions from the limited double precision words found in the Model and should require no further explanation. The input on the stack before executing the word and the output left on the stack
afterwards are indicated in the format used in the fig-FORTH GLOSSARY. You will note that several of these are mixed double and single precision operations which are sufficient for the requirements of this program.

The date compression routine is really simple. When $I$ find the time $I$ will develop an algorithm to convert the date to a true Julian day and store the least significant value. This would make calculation of the time between two given dates easy. In the meantime, the present routine allows one to enter the date as numerical values separated by slashes, a commonly used format, and reduce the value to a single 16 bit integer requiring only two bytes for storage. The routine provides an example of using a delimiter other than a space to parse 'WORD' and the use of 'NUMBER' to interpret a numerical value without searching the dictionary. After the parsing of the input, three double precision numbers are left on the stack. The word 'DATEBIT' defines a simple algorithm which is applied to reduce these three double precision values to 16 bits. The execution of '?DATE' first prompts with the format to be used, then waits for the value to be entered. The value is then converted to the 16 bits and left on the stack for starage. since '. ' is used to conote "print" in FORTH, '.DATE' is defined to print a properly formatted date from a 16 bit integer on the stack. This routine is useful as an example of conversion of a binary value to a text string for printing.

```
SCR "26, SCREDN 26: ?SAmONNT ANO. SNOONT
        \ define action tor each wele cose ' 
```



```
        MSCALI." INPUT ERAOR "CR:
        osOAlP CPA MalBRLE NSCALE - ISCNLE CFA,' 2SCACL CFA.
        - isCaLE CFA.
        I scele double crecision velum secordingto
    Scalr DpC fimIN 2. NSCALE + Execure: 
    9 I'vit for decinal value and seale it - leqvevilue on st
```



```
            \ Print d from scack as $ and right justic
            EUP ROT ROT DAES \ 1 $6 HOLD &S SIGN
            36 DMIT DUP O SWNP - SPACES TYFE;
|
CK
```

Finally, we have a Screen to define some FORTH words used to input and output dollar amounts and convert them to and from double precision 32 bit integers with
the necessary scaling for the location of the decimal point. In FORTH, the use of a decimal point forces an input number to a double precision integer which takes four bytes. A convenient FORTH primitive word, 'DPL' for decimal point locator, keeps a count on the number of digits entered following the decimal point. Utilizing this value as an input for a case type word, the numerical value entered can be scaled properly, regardless of how many digits are entered to the right of the decimal if any. This method of executing a case like routine is straight forward. First, the action to be taken in each case is defined. 'OSCALE' means that there were no digits to the right of the decimal which requires that the entered double precision integer must be multiplied by 100. In a similar manner '1SCALE' is used meaning that there was only one digit entered following the decimal point and the entered double precision integer must be multiplied by 10 . '2SCALE' does nothing since no scaling is needed. Finally, if more than 2 digits are entered an error must have been made an an appropriate error message is given. Once each of the cases is defined, their code field addresses, 'CFA', can be stored beginning with the address of a defined variable 'NSCALE' and extending into the alloted space. The word 'SCALE' then finds the value of the variable 'DPL' and counts over to the proper code field address which is then placed on the stack and the selected word is executed.

After this scaling operation, the word to input a dollar amount '?\$AMOUNT' is defined which leaves the scaled double precision integer on the stack ready to be stored. Finally, a routine defined by the word '.AMOUNT' connoting "print dollar amount" will print the double precision integer on the top of the stack as a dollar value right justified in eight spaces.

There are certainly other and probably better ways to accomplish the work done by these three Screens of utilities, but they work. The way they work provides some examples of the beauty of FORTH as it
exists in the Model.

With these five Screens, we can very quickly define a record for a data base with custom selected fields and then the associated file characteristics. In the past, $I$ have several times included in a data base values calculated from other values in the base. On occasion, it has been necessary to change one of the original values. This has always required that the calculated fields be redone, too, I now find that it is more convenient to enter only the basic data. All calculations can be made while the output is being formatted and printed with no significant loss of time. The slowest part of printing the formatted result is the delay in the output device.

```
SCR }2
    6 CR ." SCREEN 27: DEMO FILE - RECORD GENERATION*
    & DEIELO TAG ( a taq )
        30 TFIELD NAME (item name )
            2 DFIELD DAY (the date)
            4 DPIELD DOLLAR ( a dollar amount)
                2a0 ( number of recocds) 400 ( starting block)
                fILE DEMO
    INMME ( walt for name then store it in record
        NAME DROP 30 }32\mathrm{ FILL QUERY 1 TEXT PAD COINT
        NMME ROT MIN CMOVE UPCATE 
    A.NAME (pelat name fleld) NNME TYPE:
                                    ( the rest follow in the same way)
        IDAY TDATE DAY I UPDATE; : .DAY DAY Q .DATE
        : IDOLUAR PSAMOUNT DOLLAR D! UPDATE ;
    : .DOLLAR DOLLAR De .SAMOUNT ;
    15:. REC CR REC: 3.R 2 SPACES .NAME .DAY 2 SPACES .DOLLAR ;
```

As an example of the definition of a new data base, I have chosen one in which each record would be allotted 4 fields for a two byte tag, a 30 byte stock name, a two byte date, and a 4 byte stock price. Though little could be done with this as a data base, it does provide an example of each type of input. Finally, a simple set of routines is given to clear the records, input new records, and print out a list of the records in the file.

As a matter of convention, I give each field a name with no prefix. Thus, a data field name will leave an address on the stack and a text field name will leave an address and count on the stack. By using the FORTH connotations of'!' for store and '.' for print, I define some utilities for inputting data and text and printing out the respective fields. From these utilities, I can assemble an input format and an output format as desired. I have not included routines for error checking which
would be most desirable especially in a hostile environment.

Now, to examine the actual example of the definition of a file which we will =all 'DEMO'. Each record will begin with zero offset from the record address and a ' 0 ' is entered followed by '2' for a two jyte length of a data field to be named 'TAG'. Many occasions in later aanipulation of records make it desirable :o have such a field for adding flags, etc. Following this definition, the iength value of 2 is left on the stack so that for the next field, only its length need be entered. In this case, a text :ield of 30 bytes which is given the name 'NAME' which then leaves the value of 32 : the length of the 'TAG' field plus the 'NAME' field) on the stack. Then a two byte data field, 'DAY' is reserved for a 16 bit compressed date and then a four byte data field 'DOLLAR', for a double precision integer value of a dollar amount. With this, the 4 fields within the record of a new file are defined. Next, we will define the file name. According to the utility for generating a new file, we must first add to the value of the record length remaining on the stack, a value for the number of records we plan to include in the file and then the first block number to be used as determined by the FORTH implementation in use. Then, we use the word 'FILE' to create a file with these paramters and give it the name 'DEMO'. The data base file is now defined. For the record number whose value is in the variable 'REC\#', we can place the value of the address of the data fields and the address and count of the text fields on the stack by simply entering the field name. Next, a few simple utilities will make accessing these new fields easier.

Remembering the connotations associated with the FORTH words '!' and '.' we will define words to input data or text to the appropriate fields of that record whose value is currently in the variable 'REC\#'. These are simple file primitives which will then be available for routines to format input and outputs as desired.

The field 'TAG' is not used at this time and specific routines are not defined. To store a name in the name field, we define the word '!name'. This routine first fills the existing field with blanks, ASCII 32 (decimal) and then pauses for input from the keyboard. The input text is truncated to the maximum length of the text field if necesary and then moved to that field. In order to output the name in the field, we define the word '.NAME'. In a similar manner, we define '!DAY' to store a 16 bit integer value of a date which has been compressed into that field. In the earlier utilities, we have already defined '?DATE' which waits for a date to be input and leaves the compressed value on the stack. All that is necessary is to put the address of the field on the stack with 'DAY' and then store the encoded date there. We then define '.DAY' to output the date stored in the 'DAY' field. We get the 16 bit value stored there to the top of the stack and use the previously defined word '.DATE' to output it in the proper format. Finally, we define '!DOLLAR' to parse a dollar value input with a decimal point in any location and scaled to a double precision number which is then stored in the proper field. In a similar manner'. DOLiAR'is defined to format the stored double precision integer to a right justified eight digit number preceded by a dollar sign. With these definitions, we have completed a set of PORTH words to input and output data from records in our data base.

Immediately after putting data into a record, it is often desirable to see what is actually present in that record. The values in each byte of a record can be displayed using a dump routine. Simply place the desired record address on the top of the stack by entering the record number followed by our file utility word 'READ' and 'ADDRESS' followed by the length of the record and the word for your dump routine. But the byte values printed out in hex or decimal are not really all that helpful. It is hard to interpret the numerical value in their byte pattern. A convenient word '.REC' is defined to print out the current record number followed by
the formatted output of the value in each field using the above utilities and an appropriate number of spaces and carriage returns. This is the most rudimentary form of a formatted output. If desired, the output could be presented in reverse video by a slight modification of this routine. It could also be placed anywhere on the screen.

SCR 28

```
    9 CR ." SCREEN 28: DEMO FILE - CLEAR. DATA, INPUT, CUTPUT *
        (clear especially tag in the record in file)
        CLEAR.DATA READ TAG 128 g FILL UPDATE:
        ( example of formating for input)
        INPUT O READ TAG 2 1+ UPDATE DUP TAG : READ
        CR CR ." ENTER NAME \(\rightarrow\) " ! NAME
            CR :" ENTER DATE \(\rightarrow\) " IDAY ( has a format prompt)
            CR ." ENTER AMOUNT \(\rightarrow\) " IDOLUAR
            . REC FiLSH ; (sove this record on disk)
```





```
    ; 5
13
14
15
```

Finally, a few examples of formatting input and output routines are shown on the last Screen. First, it is desirable to clear all data in a file with a word 'Clear.data' before entering new data. This particular definition clears only the first block, all that is necessary in this application. You should be able to modify the definition of this word to meet the requirements of your application and par ticular implementation of FORTH.

I use the 0 record in a file for a variety of information about the file which I can address directly from the address of its first byte without using the field definitions or 1 can use specific bytes or fields in ways other than I have defined them. In this example, I use the value in the integer at the field 'TAG' in the 0 record to keep track of the last record currently in the file. When this record is cleared with 'clear.data', a value of 0 is present in the location of 'TAG' which means that there are no records present. ' 0 READ' places the value of 0 in the variable 'REC\#' and then 'TAG' places the address on the top of the stack and '@' gets that value, the last record number used in the file. To add a new record, this value is incremented and then duplicated on the stack. The top coy is stored back in the field of 'TAG' in the 0 record which is updated. Then
the second copy is placed in the variable 'RECF' and we are ready to fill in the information for the next record.

A series of prompts can be formatted on the screen in any convenient arrangement as in this example. Following the desired prompt for each field, the previously defined word is used to get the information for the field and store it there. After entering a record, it is always nice to see the data you actually put in. This is done with the word '.REC' followed by the FORTH primitive 'UPDATE' to flag the buffer as altered and 'FLUSH' to save the new record on the disk in the file. This assures that the image of the record which is displayed is the version saved on the disk.

An output format can be developed in a similar manner. In this example $I$ have included a check to see if there are any records in a file because the 'DO'... 'LOOP' will always print one loop and peculiar output is generated if the bytes in the fields are all set to zero. This output routine presents a simple list of the record numbers and the formatted content of the fields.

In conclusion, I find this approach to file definition is time saving and hope that you will find it useful. The discussion of the FORTH utilitie: used to define a new data base file and the example example of handling data provides some elaboration of the information included on the Screens. This vill be a review for one who already has learned the primitives in the FORTH Model and understands how the language works, but perhaps the discussion of these Screens will help those less experienced. There is nothing sacred about the techniques used here. Modify the various words to suit your particular needs. It is easy enough to develop new formats interactively. However, I would encourage you to utilize and build on the standards of the fig-FORTH Model. When the '79 Standards become generally available, it should be relatively easy to update your Screens without changing the format of the record file.

The importance of utilizing an accepted standard in developing programs for titimate use in a wide variety of imple－ sentations of FORTH cannot be over－ emphasized．

I wish to thank Bill Ragsdale for his encouragement to write this discussion sased on his presentation to the FORTH ：YTEREST GROUP at one of their monthly meetings last year．

APPLICATION NOTE：
These FORTH routines have been developed ：$:$ a FORTH OPERATING SYSTEM for the EaTHKIT H89．This system is available E＝om the MOUNTAIN VIEW PRESS，Box 4656， Yountain View，CA 94040．The compiled FORTH program image can be saved on disk and will be up and running in less than Esur seconds from a cold boot．The system as 1024 byte blocks which also increases ：se speed of operation．

However，after develoment，the Screens －ere loaded on a FORTH implementation ferived from the fig－FORTH FOR 8080 as SEMBLY SOURCE LISTING which is available $\therefore=0 \mathrm{~m}$ the FORTH INTEREST GROUP，Box 1105 ， ミan Carlos CA 94070，in printed form and $\equiv$ ：－eady on disk also from the MOUTAIN VIEW Eess．This version has 128 byte block ミニi operates in conjunction with CP／M．To ：－is has been added the fig－EDITOR from ：－e fig－FORTH INSTALLATION MANUAL and a single extension，DUMP，used to illustrate ：Se appearance of the records as stored in $\leq$ slock．

The printed session illustrated was made using the $C P / M$ control $P$ to echo the ＝：tput on the printer．The session starts －ith CP／M loaded and its usual prompt． Z＝e CP／M file，FORTH60．COM，is the object module of the fig－FORTH Model．The varning messages are not on Screens 4 and ；and the warning flag is turned off． Fien，the Screens for the fig－EDITOR and a Eood dump routine are loaded．Finally， ：－e Screens discussed are loaded．The E：e＇DEMO＇is called and the application ： ：some of the file utilities is illus－ ：－ated．This presentation will hopefully
assure that there are no errors in the printed Screens．

## BIBLIOGRAPHY

Brooks，F．P．，Jr．，THE MYTHICAL MAN－ MONTH，Addison－Wesley Publishing Company， 1975.
fig－FORTH INSTALLATION MANOAL，GLOSSARY， MODEL，Forth Interest Group，Box 1105，San Carlos，CA 94070.
fig－FORTH FOR 8080 ASSEMBLY SOURCE LISTING，Forth Interest Group，Box 1105， San Carlos，CA 94070.


## HELP WANTED



## fig-TREE TELECONFERENCE

> (415) 538-3580

If you are an active FORTH programmer, or just have an interest in FORTH, you will want to save this phone number. With your terminal or computer and a modem, the number will get you on-line to a dynamic data-base on FORTH.
Want to ask a question? Want to know where and when the next important FORTH Interent Group seminar, meeting, workshop, or other event is going to be? The figTree has a calendar section where you can find out about these events and let others know about yours. Want to find out about FORTH-related software, products and services?

Dial-up the fig-Tree for on-line information. Die any 300 or 110 baud modem, and type several carriage returns; then the system is self-instructing.

FORTH PROGRAMIERS Experienced with mini/micro computers and peripherals to produce new polyFORTH systems and scientific/industrial applications. Degree in science or engineering and knowledge of FORTH essential.

PRODUCT SUPPORT PROGRAMMER Responsible for maintaining existing list of software products, including the polyFORTH Operating System and Programming Language, file management options, math options and utilities and their documentation. Also provide technical support to customers.

PROJECT MANAGER Supervise applications and special systems programaing projects: writing proposals, setting technical specifications, customer liaison, hands-on prograrming, and supervision of senior programers. Extensive FORTH programing experience, some scientific or engin!ering background and management skills required. Bachelors degree or equivalent.

Contact: Min Moore FORTH, Inc. 2309 Pacific Coast Hwy. Hermosa Beach, CA 92054 (213) 372-8493

## WRITERS WANTED ANY FORTH SUBJECT SEND TO:

FORTH INTEREST GROUP
P.O. Box 1105

San Carlos, CA 94070

# INCREASING fig-FORTH DISK ACCESS SPEED 

by Michael Burton

Anyone who has used $C P / M$ and has then -sed 8080 fig-FORTH will have noticed that $\because ? / M$ is much faster than fig-FORTH when reading or writing data on floppy diskztes. The reason for this apparent speed ::fference lies in the manner in which $?$ ?/M stores its files as opposed to how : g-FORTH stores its screens. (Editor's -rte: Speed is also reflected in hardware jetails such as interleaved formatting and $\therefore$ :rect memory access. It is not necessarily a FORTH characteristic.) I shall : Sempt to explain the difference.

A single-sided $8^{\prime \prime}$ diskette formatted in :Se normal manner contains 77 tracks, with asch track containing 26 sectors with 128 :F:es of data in each sector. In order E: the disk controller to be able to find 3 particular sector in a given track, -eader data is stored on the diskette just =: 三or to each 128 byte data block - a sort : : preamble. Among other information in :-is preamble is the sector number. A Esrmat program writes this information on Each track in a consecutive manner; in :-ier words, immediately following the : siex hole pulse is sector $1,2,3, \ldots 26$.

A program that reads a sector must ::est select the proper track and proper sector, then must read that sector's data s=i store it someplace for use. It is ja:rly easy to select the proper track and sector and read the data; the problem : oees in trying to read two consecutive sectors. There is not enough time between :-e time when the first sector's data is sead and the time when the next sector is siailable, to store the data from the Erst sector and request the data from the second sector. This means that reading :onsecutive sectors 5 and 6, for example, eequires a minimum of two revolutions of :.e diskette.

CP/M accesses files faster than figFTRTH accesses screens because the files are not stored in consecutive sectors.
$C P / M$ uses a translation table to tell it which sector to use. Someone figured out that while storing the data from one sector, about five more sectors go by before $C P / M$ is able to read another sector. So instead of storing a file in sectors $1,2,3 \ldots$ it uses its translation table and stores the file in sectors 1,7, 13, etc. This means that 1024 bytes of information can be read or written in two or three revolutions of the diskette instead of eight.

What can be done about the manner in which fig-FORTH reads/writes screens? A CP/M-style translation table could be added to fig-FORTH, but that would make the diskettes, and the FORTH program, incompatible with the rest of the FORTH world. Instead, the diskettes can be formatted to look like a CP/M translation table, which is extremely easy and still allows compatibility. A diskette would look like this:

## Sector

Old format: $1 \begin{array}{lllllllll}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13\end{array}$
New format: $1 \begin{array}{llllllllllll}14 & 10 & 23 & 6 & 19 & 2 & 15 & 11 & 24 & 7 & 20 & 3\end{array}$
Old format: $1415161718192021 \quad 22 \quad 23 \quad 24 \quad 25 \quad 26$ New format: $\begin{array}{lllllllllll}16 & 12 & 25 & 8 & 21 & 4 & 17 & 13 & 26 & 9 & 22 \\ 5 & 18\end{array}$

Most format programs use an incrementing register to supply the proper sector number when formatting. To implement the translation scheme, a table must be added to the program and must be accessed in place of the sector register when formatting the diskette.

With this new format, fig-FORTH still reads 'consecutive' sectors (1, 2, 3, etc.), but they are available sooner. Using the new style format, fig-FORTH should be able to read or write a screen in two or three disk revolutions instead of eight.

Two simple tests were run to determine how this affects fig-FORTH performance:

|  | Old <br> Fownat | Nicw <br> Founat | Speed <br> Increase |
| :---: | :---: | :---: | :---: |
| LOAD 13 Screens: | $1: 28$ | $1: 17$ | $12.5 \%$ |
| COPY 50 Screens: | $2: 24$ | $: 53$ | $63.2 \%$ |

If you can't reformat your diskettes, you may choose to copy diskettes simu lating interleaveing by this program. The multiple screen copy definition used for the second test is listed in screen 167.

SCR \# 167


# MUSIC GENERATION IN FORTH 

by Michael Burton

The General Instruments programable sound generator (PSG), the AY3-8910, can be used to produce very acceptable three voice music when properly programmed. FORTH's background as a device control language makes it a good choice to use with the PSG for music production.

The programable sound generator is capable of producing sound on three separate analog channels. The amplitude and/or envelope of each of these channels is also separately controllable. Although the PSG is used by several manufacturers on their music boards, the board that was used for the development of the attached music constructs is the $S-100$ Sound Effects Board produced by Digital Research Computers of Garland, Texas. This particular board contains two AY3-8910 chips, allowing up to six voices to be generated simultaneously.

Now, for an explanation of the music screens. Screen 51 consists of definitions of $I / O$ port values and variable declarations. The variable ALV1 is the melody voice amplitude (voice one) and the variable ALV2-6 is the harmony voices amplitude (voices two through six). These amplitudes may be varied from 0 (off) through 15. It is a good idea to keep the harmony amplitude about two steps lower than the melody amplitude, in order to make the melody stand out. The variable DVAL controls the length of the notes, DVAL being the length of a whole note. The variable IVAL controls the length of the slight no-tone period between notes. Together, DVAL and IVAL control the song's tempo. Experimentation is necessary with these two variables to produce the proper tempo for a particular song.

Screen 52 contains all the definitions necessary to access the $S-100$ Sound Effects Board in order to play music. The only PSG registers currently being used in music generation are the tone period, enable and amplitude registers. Note the use of the 8080 fig-FORTH peculiar word

P!. P! simply sends a data byte to a particular $1 / 0$ port.

Screen 53 marks the start of the actual definitions used in producing a song. The definitions vl through v6 are used to select voices 1 (melody voice) through 6. These definitions do not turn the appropriate voice on, they merely select it so that a tone period (note) may be get for that voice. Voices are actually played only when a note duration is selected. The definitions vidis through vfdis are used to disable a particular voice the next time a note duration is executed. They do not turn a voice off if it is currently being played, they just turn it off the next time it is supposed :o be played.

Screens 54 and 55 define the musical zotes. The lowest note that may be played :s $b$ of octave 0 , and the highest note that may be played is $b$ of octave 8.

Screen 56 contains the definitions RINIT, VON and VOFF. RINIT initializes ail the registers on each PSG. RINIT is :he only place where the amplitude of the $\because$ oices is initialized, and should be used eefore playing any music. The definitions $\because O N$ and VOFF are used to turn all selected rices on and off. They are used inside :he note duration definitions and are not reant to be used in a song definition.

Screen 57 contains definitions for rest jurations, from a sixty-fourth rest (fr) : 0 a whole rest (wr). It also contains jefinitions for slur note durations, from a sixty-fourth slur (fs) to a whole slur ws). A slur note is one that does not go :Ef after its duration is finished, a:lowing a smooth transition between notes由en desired. Screen 57 also contains Effinitions for dotted slur durations, E=om a sixty-fourth dotted slur (fds) to a wiole dotted slur (wds).

Screen 58 is the last of the music :sastructs screens, and contains defini:Sons for note durations, from a sixtyEourth note (f) to a whole note (w). Note -iat after the note is turned off, a
slight delay (IVAL) is introduced so that the notes will be distinct from one another. Screen 58 also contains definitions for dotted notes, from a dotted sixty-fourth (fd) to dotted whole note (wd).

There is room for improvement in these music definitions. Control of the notes' envelope could be introduced to simulate other musical instruments, and restrictions imposed by the non-interrupt nature of the note duration generation could be eliminated. These exercises will be left to other aspiring composer/ programmers.


Song - Jelius Christ Supetstas

 v3 ck va as 9 vidis vidis vi ch v2 is e wi ca vi as
 va as 4 vidis vadievi ga vigs vi tis va iso vi gh v2 of v3 ef va os a vidis vadis vien va est vi ch vi gs vies va cs eb 4 vidis vidis vidis vidit vsdis v6 $\mathrm{g}^{3} \leq \mathrm{s}$.
Jc2 vadia v2 qs vi es vict vi ais c6 es vadis. va c5 vigu ct vicheatsec6e alse; gnvi vi dis vidite viois vidis vi ais qS क्य vidis
$\rightarrow$

$$
\begin{aligned}
& \text { Sonq - Jesus Christ Superstar }
\end{aligned}
$$

$$
\begin{aligned}
& \text { vi bs vi tj vica bseck. bs evick v2dis }
\end{aligned}
$$

$$
\begin{aligned}
& \text { v1 ais. Ch ent en ed dis v2 ats vi g5 va } \\
& \text { dic fa gh ss vidis vidis vadis sit th v2 w wa }
\end{aligned}
$$

## OPTIMIZING DICTIONARY SEARCHES

Paul van der Eijk<br>5480 Wisconsin Avenue, \#1128<br>Chevy Chase, MD 20015<br>(301) 656-2772

Recently, I finished the implementation of fig-FORTH on my Radio Shack model II. I must admit that I did not follow the FIG model precisely; some high level definitions were recoded in assembler to increase their speed. For example, sign extraction in the divide and multiply words gives an execution time improvement of a factor two. These improvements are predictable and probably implemented many :imes already.

One deviation from the FIG model I want to share with you is the structure of the dictionary.

In the FIG model, the Link Field Address is stored after the last character of the name. When (FIND) searches the dictionary for an entry, the lengths of the strings are compared. If the comparison fails, and this happens a lot, the sharacters stored are scanned for a high jit in the last character. When the scan stops at the last character, we know the address of the LFA, because it follows the -ast character. It will be obvious that the time spent on searching for the LFA will be linear with the average numbers of characters stored for an entry. One way to get around scanning is adding an additional byte in every dictionary entry, indicating the actual number of characters stored. Another approach was taken by Robert Smith, see FORTH DIMENSIONS, Vol. $\therefore$ No. 5 .

The structure I implemented puts the LFA in front of the Name Field Address. When (FIND) stores the address of the NFA in a machine register, a search for the -FA is not necessary because it precedes the NFA directly. In addition, the characters of the entry can be stored in normal order, which makes changing ID. unnecessary.

The new dictionary structure can improve compilation speed substantially.

An application program 70 screens long took 210 seconds to compile; the new dictionary structure reduced the compilation time to 98 seconds.

To implement the new dictionary structure, the following words have to be rewritten:

CREATE VOCABULARY LFA NFA PFA . (FIND) has to be rewritten as well, but is not given here because it is machine dependent.

```
( LFA preceede n%a 1 of 2:
MEX
: CREATE -FIND IF DROP nFA 1D. 4 mESSACE SPACE tILEN
                                    ( check unlque in CUQRENT and CONTEXT)
        HERE CE WIDTIL MIN DR ( save number of chare stored)
        HERE OAD TOCGLE IIERE R * DUP 080 TOGGLL
        Dup 2+ l 1+ -CMove (amonge and deliniter bits)
    LATEST HERE ( Gove entry doun to insert lfa)
    LATEST HERE 1 HERE 2+ CURREMT & I
    R) 3 + ALLOT HERE 2+ : :
    : NFA 3 - -J TRAVERSE
```



```
#
( LfA precceda Hfa 2 of 2: Paul van der Eijk aprll-12-1981)
: vocabulary <billus ciarert e 2+ . 0aOBl .
    HERE VOC-LINK O , VOC-LIMR I
    DOES) CONTEXT 1;
( the following chonge in -Find smeda up dictionary searche*
(in came the CURRENT and CONTEXT vocabularles are the ame.
(the change te not necsesery for the new dictsonary structure)
    : -TIND &L WORD LERE COUTEXT & (FIND) DUP O-
        IF DROP HTEST CONTEXT O OVER -
            IF IIERE SWAP (FIUD)
            ELSE DROF O
    then tien;
DECIMAL ;s'
```


## MEETING

## POTOMAC FORTH INTEREST GROUP MEETING

Program was presented by Paul van der Eijk on IQS - An Interactive Query System. He described this system which lets the user create, edit, search and list a file without writing a program.

The next meeting is Tuesday, Aug. 4, 1981 at Lee Center, Lee Hiahway at Lexington St., Arlington, Virginia from 7:00-9:00pm.

# TRACING COLON-DEFINITIONS 

Paul van der Eijk<br>5480 Wisconsin Avenue, \$1128<br>Chevy Chase, MD 20015<br>(301) 656-2772

This short article describes a few simple words to trace colon definitions. When $I$ am completely lost trying to find a bug in a FORTH program, I use colon tracing to get a print-out of all words executed together with a few parameters on the data-stack. Such a print-out is often enough to spot the bug; in addition, it gives some insight how many times certain words are executed which can help to improve the execution time of a program.

How it works:
A technique to trace colon definitions is to insert a tracing word directly after the colon.
i.e., : TEST T1 T2 ; TEST can be traced by having a definition compiled as if it were:

```
: TEST (TRACE) T1 T2;
```

When (TRACE) executes, the address of the word following it is on the return stack. Subtracting two from this address will give the parameter field address, from which we can reach the name field address using the word NFA. In order to enable/disable the trace ouput, the variable TFLAG is used; a non-zero value will enable the output and a zero value will suppress the trace output.

The insertion of the (TRACE) word can be automated if we redefine the definition of the colon.

The colon is redefined to insert the runtime procedure for the colon followed by the address of (TRACE).

Note that the address of the colon runtime procedure is obtained by taking it from the code field address of the word (TRACE).

## Improvements:

1. If we save in (TRACE) the value of the variable OUT and direct output to the line-printer, words doing formatted terminal output can be debugged effectively.
2. A variable TRACE is introduced to control the insertion of the word (TRACE) in the new definition for the colon.

If the value of TRACE equals zero, (TRACE) is not inserted, if the value is non-zero (TRACE) will be ingerted.

This enables tracing code to be inserted in a selective way by changing the value of TRACE preceding a colon definition.
i.e.:

0 TRACE ! : TEST1 Tll Tl2 ; ( TESTl will not be traced)

1 TRACE ! : TEST2 T21 T22 ; ( TEST 2 can be traced)

```
U (trace culon worde: Paul van der E1jk aprll-12-1981)
PORTII DEPINITIONS
O vaRIa⿱LE TTLAC
                (conerola trace output)
(tive trace output. to be inmerted at firtt mord
        TFLAC ( ( trece output if non-rero
        IF CR R 2 - MFA DUP 10. (back to PFA NFA for name
        31 AMD 32 suAp - SPACES (add apecee to name)
            2 4 DO spe I + 8 .N -2 +LOOP (ahow atack)
        THEN ;
    : ( redefined so ingert trace word after colon)
        qExEC ICSP CuRENT e context i enEATE
```



```
    ( exmple: crace folloving uae of I and C!)
: l Cl Cti
                                    ;s
```


## meeting

## NEW YORK CHAPTER

First meeting of the New York Chapter was held on June 23, 1981. There were five FIG members and one non-FIG person in attendance. The second meeting is scheduled for August 25, 1981 and subsequent meetings every other month.

## FORTH, Inc. NEWS

## - re Growing

FORTH, Inc. expects to double its staff -ithin the next year to accomodate .acreased product demand and applications :-ograming. (See job openings listed $\doteqdot$ sewhere in this publication.)

The latest addition to our staff is ::ogrammer Charles Curley. Curley is a Ermer freelance writer and programmer who Eits and publishes the Ohio Scientific sers' Newsletter.
"I put FIG FORTH up in my own Ohio :sientific C2-8P DF and liked it," he : mments, "but I wanted to learn FORTH sistematically, and I figured this was the :Est place to do that. At FORTH, I get :aid to do what I like to do."
:-ner News
President Elizabeth D. Rather was a peaber of a panel on programing languages $\therefore=$ small computers at the NCC Conven: : on. She was featured in both Computer--i: 1 d and Computer Business News.

Programmer Mike LaManna has relocated $:=$ Long Island, New York, and is working $\therefore$ the 68000 polyFORTH. It should be ::ailable midsummer.

## E: : yFORTH Palo Alto Users Groups Starting

Dr. C. H. Ting has volunteered to Chair - Pe Palo Alto Thread of the FORTH Users :=oup for the first three months. Anyone $\therefore$ :erested in joining the Users Group may : :ntact Dr. Ting at Lockheed Missiles and三jace Corp., (408) 742-1101 or Al Krever : = FORTH, Inc. (213) 372-8493.

## ミesent Applications

FORTH, Inc. has produced a computer -zperical control program for $L \& F$

Industries' rotating longitudinal-stretch forming machine. This 80-foot-long, three-story-high giant weighs over a million pounds and pulls 750 tons. It is used to form, stretch, bend and stretch wrap aluminum, steel and titanium sheet metal or extrusion parts (typically panels used in C5A-sized aircraft).

An LSI-11 detects the pield point of the metal and maintains a pre-set stress as the operator directs the initial operation; it then takes over full control and manufactures identical production parts. This computer program, written in poly FORTH, coordinates the motion of nine simultaneously moving servo-controlled axes with a resolution of .008". The system also displays on a CRT the position of all axes and a graph of the stress curve showing the yield point of the metal. Mike La Manna, Jim Dewey and Gary Friedlander were involved in the project.

## Starting FORTH Preprints Available

A few unsigned preprints of starting FORTH are available now for $\$ 30$ (plus 6\% state tax). The Prentice-Hall edition will be available in book stores September 8. To order a preprint, send a check to Winnie Shows at FORTH, Inc., 2309 Pacific Coast Hwy., Hermosa Beach, CA 90254 or you may call her at (213) 372-8493 with a VISA or MASTERCHARGE number.

## FORTH, Inc. Seminars, Workshops, Classes

| Location | Seminar | Workshop |
| :--- | :--- | :--- |
| Chirago | August 4 | August 5 |
| Boston | August 6 | August 7 |
| Boulder, CO | September 1 | September 4 |
| Los Angeles | October 15 | October 16 |
| San Diego | October 22 | October 23 |

An introductory class in polyFORTH programming will be offered August 10-14 at FORTH, Inc. Call Rris Cramer for details (213) 372-8493.

## FORTH VENDORS



Morth Star
The Software Works, Inc. P. O. Box 4386

Mountain View, CA 94040
(408) 736-4938

## 081

Consumer Computers
8907 Lakese Blvd.
LaMesa, CA 92041
(714) 698-8088

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

Technical Producte Co.
P. O. Box 12983

Gainaville, FL 32604
(904) 372-8439

Tom 2 i maer
292 Falcato Dr .
Milpitas, CA 95035
600066809
Kenyon Microsystems
3350 Walnut Bivd.
Houston, TX 77042
(713) 978-6933

20P-11
Laboratory Software Syatean, Inc.
3634 Mandeville Canyon Rd.
Los Angelee, CA 90049
(213) 472-6995

John S. James
P. O. Box 348

Berkeley, CA 94701
Ths-60
Miller Microcomputer Services
61 Lake Shore Rd.
Natick, MA 01760
(617) 653-6136

The Software Farm
P. O. Box 2304

Reston, VA 22090
Sirius Syateas
7528 Oak Ridge Iny.
Knoxville, IN 37921
(615) 693-6583
rint
Eric C. Rehnke
540 S. Ranch View Circle $\$ 61$
Anaheim Hille, CA 92087

## 8000/250/CP/B

Laboratory Microsyateas
4147 Beethoven St.
Las Angeles, CA 90066
(213) 390-9292

Mitchell E. Timin Engineering Co.
9575 Genesse Ave. fe-2
San Diego, CA 92121
(714) 455-9008

Coneulcant
Henry Laxen
1259 Cornell
Berkeley, CA 94706
(415) 525-8582

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

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

Pinuare, Boarde and Machimes Datricon
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 3669

Anaheim, CA 92803
(714) 632-2862

Zendex Corp.
6398 Dougherty Rd.
Dublin, CA 94566
Variety of Fonts Producta
Interactive Computer System, Inc.
6403 Di Marco Rd.
Tampa, PL 33614
Mountain View Preas
P. O. Box 4656

Mountain View, CA 94040
(415) 961-4103

Supermoft Aasociates
P.O. Box 1628

Champaign, IL 61820
(217) 359-2112

Consultant
Creative Solutions, Ine.
4801 Randolph Rd.
Rockville, MD 20852
Dave Boulton
581 Oakridge Dr.
Redwood City, CA 94062
(415) 368-3237

Elaer W. Fittery
110 Mc Gregor Avenue
Mt. Arlington, NJ 07856
(213) 663-1580

## Go PORTH

504 Lakemead Hay
Redwood City, CA 94062
(415) 366-6124

Inaer Access
P. O. Box 888

Belmont, CA 94002
(415) 591-8295

## DECOMPILER FOR SYN-FORTH

SYN-1 User': Group<br>PO Box 315<br>Chico, CA 95927

The following decompiler works very vell except that because INTERPRET is not =emembered by ;S nor (;CODE-) nor QUIT, Sis FORTH decompiles loop !

```
scan1 (ist jlia
n( scempazR...) wex
? vocamlary util momolate mortw delimitiows
    : PIC: 2 - spe + ; GTIL seflnitions
```




```
        {(*)
            guIT TEST-
```



```
is --
sce:?
```




```
        clse:0; complle TST- ir mot tur e 2% rea 80. 2t grace
```




```
        elst O : lit TEST= DuF If max TMeN
            MEmch 75%0
            onanmca ttst:
```



```
            muen tuen tuem becmal mem;
        is}
            ..occomelim... ) montw berimitiove nex
        becourluc ( ma -> mon or wetr woad) UTLL
            REX OUP 4, mup cra e; .e dectma
            DuP CPa & 803.
            If cla = Dup wima it. seace mup
```



```
                mile 10. TEStury
                mile imapmen
```



```
            clse ifa compmats.
            tuen lua cm ca ;
```



```
tworles
            becompilv.all
```





```
            150 co3, is ovte ovin on om onavicin -12 is
```





```
            * Pac nLD I is
            spaces o max -Dup omamce 12 O (00) arace (LON) at is
```






```
                    *
                    * %or
```

- count befinitions becomille pon sm-1

HE ; : wow. To.secontile beconfile

outrut : MA EFA.Contivir : mord.to.becont
oispucerext are in der palt (danich. onanicu. Lit....)


sustin ir ... Lise ... asc. ...
secompilif vailanle, wocabulaky. ....


## ENGLISH FORTH APPLICATION

Golden River company has been using FORTH
for the RCA 1802 for the last three years,
to fill a need for a low cost development
and prototyping tool with potential for
being used at remote sites where power is
not easily available.
The most interesting concept in the equipment is it uses 32 K of dynamic RAM as storage apace for up to 30 screens of source FORTH code. The equipment is designed with low power in mind and is normally used like an electric car-it is usually kept connected to an AC source, although it has nine-days battery life and can be used remotely.
The product is currently being shipped in Europe and will be introduced in the U.S. market through Golden River Corporation, 7315 Reddfield Court, Falls Church, VA 22043.

For more information, contact:
Golden River Company, Ltd. Churchill Road
Bicester, Oxfordshire 0X6 7XT
England
Phone: Bicester (08692) 44551
Telex: 83147 VIAOR G 'GRIVER'

GET READY!
FORML's COMING!

## NEW PRODUCT ANNOUNCEMENT FORMAT

In the interests of comparison uniformity and completeness of data in new product announcements, FORTH DIMENSIONS requests that all future new product announcements use the following format:

1. Vendor Name (company)
2. Vendor mailing address
3. Vendor street address if PO Box. Used as mailing address. For reference file.
4. Vendor area code and telephone number
5. Person to contact
6. Product name
7. Brief description of product uses/features
8. List of extras included (editor, assembler, data base, games, etc.)
9. List of machines product runs on
10. Memory requirements
11. Number of pages in manual
12. Tell what manual covers
13. Indicate whether or not manual is available for separate purchase
14. If manual is available, indicate separate purchase price and whether or not manual price is credited towards later purchase
15. Form product is shipped in (must be diskette or ROM-no RAM only or tape systems)
16. Approximate number of product shipments to date (product must have
active installations as of writing-no unreleased products)
17. Product price
18. What price includes (shipping, tax, etc.)
19. Vendor warranties, post sale support, etc.
20. Order turn-around time

## MEETINGS/EVENTS ANNOUNCEMENT FORMAT

In order to have uniformity and insure complete information in all meeting and special event announcements, FORTH DIMENSIONS requests that you use the following format:

1. WHO is holding the event (organization, club, etc.)
2. WHAT is being held (describe activity, speakers' names, etc.)
3. WHEN is it being held (days, times, etc.; please indicate if it is a repetitive event--monthly meeting etc.)
4. WHERE is it being held (be as complete as possible-room number, etc.)
5. WHY is it being held (purpose, objectives, etc.)
6. REMARRS and SPECIAL NOTES (is there a fee, are meals/refreshements being provided, dress, tools, special requirements, pre-requisites, etc.)

## 7. PERSON TO CONTACT

8. PHONE NUMBER/ADDRESS (include area codes, times to call and give work and home numbers in case we need clarification)


# 1981 FORML CONFERENCE 

Asilomar, California November 25-27, 1981

ATTENDEE REGISTRATION FORM

Conference Purpose:

Attendee Selection Priority:

Registration Form, Complete and return to: FORML
PO Box 51351 Palo Alto, CA 94303

## Accommodations

 Desired:The 1981 FORML (FORTH Modification Laboratory) is an advanced seminar for the presentation of FORTH language papers and discussions. It is not intended for beginning or casual FORTH programers.

The FORML Conference is limited to 60 FORTH programmers (approx. 30 family and other non-participants accommodations are also available). The priority for selection of attendees is:
lst - Paper presentors who send in their 100 -word abstract by the deadline of September 1, 1981.*

2nd - Poster presentors who send in their 100 -word abstract by the deadline of September 1, 1981.*

3rd - FORTH programmers who wish to attend only. Depending upon the response of paper and poster sessions there may or may not be room for non-presentors.*
*The FORML Conference Referees will make the final decisions on paper/poster presentors which will in effect determine attendance and priority positions.

NAME $\qquad$
ADDRESS


1 expect to: $\qquad$ present a paper, $\qquad$ present a poster session chair a section, $\qquad$ non-presentor
My topic will be: $\qquad$
Rooms at Asilomar include meals (including a huge Thanksgiving) and the price of the Proceedings is included in participant costs.
$\qquad$ Myself $\qquad$ Double \$1i0.00 $\qquad$ Single \$150.00

[^1]
# LATE NEWS 

BURKLUND \& ASSOCIATES
3903 Carolyn Ave
Fairfax, VA 22031
(703) 273-5663

Mr. Roy C. Martens
June 29, 1981
Forth Interest Group P.O. Box 1105

San Carlos, CA 94070
Sirs:
Tsk, Tsk, Tsk! You really did it this time! Tsk, Tsk, Tsk. The product review of Timin Eng. version of FORTH is totally beside the point...what you did, was crucify one of the finest versions of FORTH currrently available on the market, namely the Laboratory Microsystems, Z-80 FORTH.

I bought Laboratory Microsystems 2-80 Forth about 3 months ago, and was ecstatic with what I had bought for a paltry $\$ 50$. When I read your product review, I tried the same definitions on my 4 MHz system and found that all times were approximately 2-5\% less than your comparative data for Timins 8080 version...therefore with the differing CPU clock rates of 4 MHz for my and 6 Mhz for Timins systems on which the Laboratory Microsystems 2-80 versions were compared (how convenient is was tried on Mr. Timins systems) the $2-80$ version should reflect benchmark times approaching $30 \%$ better than those cited in the comparison test. I would have thought that FORTH DIMENSIONS would have staff expertise of a bit higher quality than that reflected in the product review article.

As for the tip-toeing disclaimers via the Editors Comment... hey, it just won't wash!

I think that FORTH DIMENSIONS owes a very large apology to LABORATORY MICROSYSTEMS, and at least a full page of space to try to counter the damage you have done to $z-80$ FORTH; or will you allow the old adage that "the truth never catches up to the lie", prevail? FORTH DIMENSIONS...Shame! Shame! Shame!


Glenn A. Burklund
Eublisher's Coment: The following letter was received in feference to a Product ReTiew by C. H. Ting in FORTH DINENSIONS, III/1, page 11-12, which compared some bench earks between CP/M FORTH from Timin Engineering and Z-80 FORTH from Laboratory Microsystems. We are printing this letter in its entirety for several reasons: to correct ayy unintentional damage to Laboratory Microsystems; to ask our members whether they jesire comparisions between FORTH and other languages and between FORTH products; if تe are to do comparisions then it will have to be by volunteers since we do not have a staff, it then becomes a problem of who and how. Any volunteers?

How to form a FIG Chapter:

1. You decide on a time and place for the first meeting in your area. (Allow at least 8 weeks for steps 2 and 3.)
2. Send FIG a meeting announcement on one side of $8-1 / 2 \times 11$ paper (one copy is enough). Also send list of ZIP numbers that you want mailed to (use first three digits. if it works for you).
3. FIG will print, address and mail to members with the ZIP's you want from San Carlos, CA.
4. When you've had your first meeting with 5 or more attendees then FIG will provide you with names in your area. You have to tell us when you have 5 or more.

Northern California
4th Sat FIG Monthly Meeting, 1:00 p.m.,

Hayward, CA. FORML Workshop at
10:00 a.m.

Orange County
3rd Sat FIG Meeting, 12:00 noon, Fullerton Savings, 18020 Brockhorst, Fountain Valley, CA. (714) 8962016.

San Diego
Thur FIG Meeting, 12:00 noon. Call Guy Kelly, (714) 268-3100, x 4784 for site.

## Northwest

Seattle Chuck Pliske or Dwight Vandenburg, (206) 542-8370.
Oregon
2nd Sat
FIG Meeting, 1:00 pm, Computers \& Things, 3460 SW 185th ' $D$ ", Alohs, Eric Smith, (503) 642-1234.

## New England

Boston
1st Wed FIG Meeting, 7:00 p.m., Mitre Corp., Cafeteria, Bedford, MA. Call Bob Demrow, (617) 389-6400, $x 198$.

Boston
3rd Wed
MMSFORTH Users Group, 7:00 p.m., Cochituate, MA. Call Dick Miller, (617) 653-6136 for site.

Southwest
Phoenix Peter Bates at (602) 996-8398.
Tulsa
3rd Tues FIG Meeting, 7:30 p.m., The Computer Store, 4343 So. Peoria, Tulsa, OR. Call Bob Giles, (918) 599-9304 or Art Gorski, (918) 743-0113.

Texas Jeff Lewis, (713) 719-3320 or John Earls, (214) 661-2928 or Dwayne Gustaus, (817) 3876976. John Hastings (512) 8351918.

Mid Atlantic
Potomac Paul van der Eijk, (703) 3547443 or Joel Shprentz, (703) 437-9218.

New York Tom Jung, (212) 746-4062.

## Midwest

Detroit Dean Vieau, (313) 493-5105.
Foreign
England FORTH Interest Group, c/o 38, Worsley Road, Frimley, Camberley, Surrey, GUl6 5AU, England

Japan FORTH Interest Group, Baba-bldg. 8F, 3-23-8, Nishi-Shimbashi, Minato-ku, Toyko, 105 Japan.

Canada
Quebec
Gilles Paillard, (418) 871-1960 or 643-2561.

## West Germany

Wolf Gervert, Roter Hahn 29, D-2 Hamburg 72, West Germany, (040) 644-3985.


FORTH DIITETSIIOTS


FORTH INTEREST GROUP
P.O. Box 1105

Volume III

San Carlos, CA 94070

## IITBIIIF

$\qquad$


74 FORTH-79 Standard-A Tool Box?

78 FORTH Engine

80
FORTH, Inc. Line Editor


96
Recursion and the
89 Ackermann Function

91 FORTH, Inc. News

## 92

$\qquad$ Marketing Column
73 FORTH Standards Corner

## FDATH OITIETSMDIS

Published by Forth Interest Group

Volume III No. 3
Publlsher
Editor
September/October 1881

Editorial Review Board
Bill Ragsdale
Dave Boulton
Kim Harris
John James
Dave Kilbridge
Henry Laxen
George Maverick
Bob Smith
John Bumgarner
Bob Berkey
FORTH DIMENSIONS solicits editorial material, comments and letters. No responsibility is assumed for accuracy of material submitted. ALL MATERIAL PUBLISHED BY THE FORTH INTEREST GROUP IS IN THE PUBLIC DOMAIN. Information in FORTH DIMENSIONS may be reproduced with credit given to the author and the Forth Interest Group.

Subscription to FORTH DIMENSIONS is tree with membership in the Forth Interest Group at $\$ 15.00$ per year ( $\$ 27.00$ foreign air). For membership. change of address and/or to submit material, the addreas is:

Forth Interest Group
P.O. Box 1105

San Carlos, CA 94070

## HISTORICAL PERSPECTIVE

FORTH was created by Mr. Charies H. Moore in 1969 at the National Radio Aatronomy Obeervatory. Charlottesville, VA. It was created out of diseatisfaction wit. available programming tools, especially for obeervatory 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.

## EDITOR'S COLUMN

This month introduces the long-promised MARKETING COLUMN. Considering that one of the best ways to proliferate FORTH is to sell it in the form of applications, I believe this column will contribute to the financial welfare of our members and help make the FORTH language a force in the software world. Questions related to all phases of marketing from product research and development to pricing, advertising and distribution channels are welcome.

Our next edition will be devoted to the conference at the University of Rochester and will be coordinated by Mr. Larry Forsely of that institution. One of my main goals as editor has been to "de-Californize" FORTH DIMENSIONS and make it reflect more accurately the opinions and techniques employed throughout the FORTH world. This next edition is a bold experiment in this regard and I have every confidence in Mr. Forsely helping to make it a successful one.

The issue following ROCHESTER wIII be devoted to mustc, graphics and games. Currently, this type of application is in very short supply and 1 am appealing to our members to submit them as soon as possible. Please remember, YOU DON'T HAVE TO BE A WRITER-our ataff will help provide whatever is necessary to make your article or game publishable-but the raw ideas and code have to come from you. Also, we are not necessarily looking for lengthy, complex and elegant applications-simple, instructive, short codes often are best and the most useful.

Pleese contribute whatever you can-ultimately the quality and utility of FORTH DIMENSIONS comes from our members!
C. J. Street

Editor

## PUBLISHER'S COLUMN

Nov. 2-4: Mini/Micro Show. Anahelm, CA
Nov. 25-27: FORML Conference, Pacific Grove, CA
Nov. 28: FIG National Convention, Santa Clara, CA
Mar. 19-21: Computer Faire, San Francisco, CA
FORTH vendors-these shows can be helpful to you in several ways. First, if you will send FIG approximately 500 flyers, $81 / 2 \times 11$, about your products, we'll display them at all four places. Second, you should exhibit at the FIG Convention on November 28 at the Marriott Hotel, Santa Clara, CA. An $8^{\prime}$ table is only \$50.00-send a check to FIG, today. Third, FIG has a prime location at Computer Faire, March 19-21, 1982 In San Francisco. We have booths 1343C and 1442C: these face the central booth area and form an island with eight other booths. Six of these booths are currently avaliable. Lets get all FORTH vendors together. All you have to do is call Computer Faire (415) $851-7075$ and tell them you want to be in the same Island as the FORTH Interest Group.

## Roy Martens

## Dear Fig:

Let me introduce myself: I'm Jim Gerow, an avid MMS FORTH user, a FORTH programmer (installer) for larger machines, and a member of the local MMS FORTH user group.

I've been referred to you by Dick Miller of MMS as a correspondent. Please let me know how I can be of service and how our MMS FORTH User Group can support you.

Jim Gerow<br>1630 Worcester Rd., ${ }^{\text {F630C }}$<br>Framingham, MA 01701

Thank you--we would appreciate any articles, ideas, bug fixes, or usable programs or tools you can send in. Looking forward to hearing from you-ed.

## Dear Fig:

San Diego has a FORTH Interest Group that meets informally each Thursday and somewhat formally the 4th Tuesday of each month.

Because of the different machines, MPU's, and operating systems, (i.e., Disk or Cassette, etc.) we have a problem of software exchange (transportability). We are considering the development of a communications package involving RS-232 modems and software.

We are interested in finding out what FIG has, if anything, in standardizing any of the communications such as protocol or hardware specification.

Currently, most of the software exchange involves hardcopy. If you have any information relative to communications between FORTH operating machines or protocol standards used, we would appreciate your help.

K. G. Busch<br>Rancho Bernardo<br>12615 Higa Place<br>San Diego, CA 92128

O.K. members-how about giving Ken a hand? Or better yet, send the info to FORTH DIMENSIONS and we will publish it for all of our members' use.--ed.

Dear Fig:
Would you please forward me a writer's kit? ['m thinking about writing something for Nov./Dec. GRAPHC/MUSIC. I have implemented a set of graphic words for Columbia Data Products' MX-964 (Z-80 Micro-*2, 512×256 bit mapped, $\$ 10-8080$-figFORTH), and am working on some musical words for a dual Gr's sound chip board. If I can get my hands on Digitalk (National Semi) early enough, maybe some work also can be done on that before the date I send out my articles.

Since you share the very same address as the 8080 Renovation Project, would you please forward the following page to them for me? Many thanks. Happy FORTH!

> Tim Huang
> 9529 N.E. Gertz Circle Portland, OR 97211

Thanks, Tim-we'll be in touch. The graphics issue is approaching rapidly (deadline is Oct. 15, 1981)-word to the wise-ed.

Dear Fig:
This is a note authorizing you to use the cartoon-style illustrations in the book Starting FORTH by Leo Brodie of FORTH, Inc. The credit line should read Leo Brodie, FORTH, Inc., Starting FORTH, a forthcoming PrenticeHall publication. Reprinting oy permission of Prentice-Hall, Inc., Englewood Cliffs, N.J. After October 1, 1981 you can leave the "forthcoming" out since the book will be in print. Thank you for your interest.

> Jim F. Fegen, Jr. Editor, The Computing Sciences PRENTICE-HALL. Englewood Cliffs, NJ 07632

Thank you Prentice-Hall. Watch for cartoons from this important work.-ed.

Dear Fig:
Here is your complimentary copy of Starting FORTH. We at FORTH, Inc. hope you enjoy it as much as Mark Garetz of INFOWORLD, who said it was the best beginner's book he'd seen.

The hard- and soft-bound editions by Prentice-Hall will be on the shelves Sept. 8.

Let us know what you think of the book. We are anxious to hear your comments.

Winnie Shows<br>Public Relations<br>FORTH, lnc.<br>2309 Pacific Coast Highway<br>Hermosa Beach, CA 90254

Thanks, Winnie. Please note the review in this issue.--ed.

## Dear Fig:

I live in a country town in Australia and the number of local computer hobbyists can be counted on one hand. I have so far converted one friend to FORTH and we have found all the back issues of FORTH DIMENSIONS very helpful with programming examples. I have had my system for about 5 years; it is an $\mathrm{Sl00} \mathrm{Z}-80$ system with recently added dual Micropolis Mod II disks. I have rewritten the 8080 FIG FORTH CP/M interface to work with Micropolis DOS and am currently reworking some of the 8080 CODE definitions to use $\mathrm{Z}-80$ instructions where they will improve the code. I am interested in corresponding with other FIG users, particularly those with systems similar to mine.

I wish to make a comment about the naming of words related to 32 -bit integer operations. The present mixture of prefixed "D" and "2" make these words more frustrating to learn and use. That would not be the case with consistent prefix character. I think that the prefix character should be "D" for double. I am sure that most of us find the prefix letter "C" easy to use for 8 bit operations and I am glad the ASCII did not allow 1/2@ to be used. When floating point comes around (for example, in a 6 byte format), it seems most likely that F! will be used, not 3!. So let's be consistent and leave digits for numbers and use a prefix letter mnemonic to indicate stack operations, etc. that are not the usual one word ( 16 bits).

Could someone please explain what the HEX value A081 is for, in the definition of VOCABULARY? I can't work it out.
<eep up the good work with the magazine.
Bill Miles
PO Box 225
Red Cliffs
Victoria 3496
Australia
Thanks for your comments, Bill. Glad to EE- FORTH is alive and well in the land down res! How about some of our members =responding with Bill and helping him over the ت,
=三-Fig:
-JRTH DIMENSIONS has grown increasingly Pa to me in the past few months as 1 have fre..y begun to "get the hang of" FORTH. I E.e running on my TRS-80 several versions of $\equiv-:=2$, FORTRAN, PASCAL, APL, SAM76 and $-\bar{F}$ under both TRSDOS and CDOS; but I have F.e: found any language harder to learn that $=\mathrm{F}-\mathrm{H}$. Part of the problem is the scope of I- -H: at the same time I'm trying to reerstand the interpreter, compiler, OS, and a artex as difficult as LISP's. I have found all - e struction manuals so far to make a drastic "- from simple concepts like $22+$, and $7 \mathrm{C} . . \mathrm{LOOPS}$ to discussions of the Dictionary F: $2 f$ Defining Words. (I think the writers had F same problems I have, of separating the - zus functions of the system.) One of the Ler. velps I received was Mr. Bumgarner's Stack Zaç:am in this year's March issue of BYTE. Te necessity of being able to visualize the fizack cannot be overemphasized. Once I was IIE: : : o do that, I starting learning in earnest.
-eving leaped this hurdle, 1 found FORTH -re rewarding than any other language to it- - One of its greatest advantages to me as (IT Epplications programmer is its (almost) -r-iv consistent syntax: operators, functions, ax: "orocedures" disappear and all you have are was that get their arguments off the stack are slace their return values on the stack. All F. applications so far have been in BASIC. - F . are as "structured" as I can make them L_z.. subroutines calling subroutines), but it nor jecomes hard to remember what "GOSUB -rol does and which variables have to have $r$ : values in order to do whatever to whom : $:$ - where' Not so with FORTH: although a
restricted use of variables does make the program somewhat less readable, keeping most of the arguments and returned values on the stack actually makes it more "writeable" because there is no need to remember what the formal or actual parameters are. Right now, because of my limited experience with FORTH, it takes longer to write a "routine" than it would in BASIC: but already the total time to test and debug is far less.

I'm using Miller Microcomputer Services' top-notch MMSFORTH, and I have absolutely nothing bad to say about these people. Last summer I drove down to New England in order to pick up some hardware, and decided to drop in on Richard and Jill Miller in Natick, MA. They showed me the utmost in hospitality, helping me purchase equipment and wasting their time in general to make sure that my trip was worthwhile. Their product is excellent: worth it at twice the price and more (you didn't hear that, Dick!)-with standard features such as Strings, Double-Precision, Graphics, a good Screen Editor, and not one, but several fine demonstrations programs. A++ for MMSFORTH.

Morningstar is a software house in southern Ontario that does mainly custom programming. All of it so far has been BASIC, but we expect to have fully switched to FORTH by the end of 12 months, D.V. No other language would have compelled us to give up "Tandy Compatibility," but the advantages of FORTH far outweigh any extra cost for the lanage.

Thanks for your attention.

> Vincent Otten MORNINGSTAR 225 Dundas St. Woodstock, Ontario CANADA N4S 1 A8

I am sure Dick Miller appreciates your comments. You might also look into Mr. Leo Brodie's new book Starting FORTH (reviewed elsewhere in this issue) that will be available in mid-September.--ed

## Dear Fig:

I very much enjoyed my first pass through your article "Compiler Security" in FORTH OIMENSIONS III/1. I plan to re-read the article when I have more time.

In terms of the multi-user environment, haven't you almost answered your own question of security always versus security on demand with your parenthetical "and the other users" remark? This was near the end, in the discussion of the possible use of a "Novice Programmer Protection" package. In a singleuser environment, more liberties can be taken, but I know I'm a novice user, having only been involved with computers since 1958 or 80 , and having only "FORTHed" non-intensively for about 3 years. My single-user system would always include the protection package (well. . .almost always). I would not, however, object to making security optional in the singleuser case (but I am not a prospect for a FORTH implementation without it).

I don't agree with your characterization of Assembler security as inappropriate. It is the ability to have unstructured code that causes many of the problems with assembly code. If it is so easy as to be tempting, we will all err. FORTH makes the cost-to-fix versus time-error-found curve perhaps less steep, but early error detection is still cheaper, and software is still the largest part of the system cost (and getting larger). I cannot argue against being able to defeat Assembler security fairly easily, however, since there may be situations in which the risk is worth it.

John W. Baxter<br>Sr. Principal Programmer Analyst<br>NCR CORPORATION<br>Coronado, CA

MR. SHAW REPLIES:
I hope that after three years of FORTH programming that you have developed good FORTH style. This should be the case unless you have let your previous 20 years of experience interfere with your learning of FORTH's simplistic concepts.

In aither case, you should be aware that good FORTH code is well thought out and very short. Most definitions, in either high level or assembler, should be very short; not more than a few lines. In very few instances is high level code ever longer. Those definitions that are long should be so well scrutinized as to the reasons for their length that the type of errors that the current compiler security would trap should not exist. Assembler code should only be used when speed is a critical factor. And then, structured code may not be the easiest or fastest to program without error, or the fastest to execute. The programmer may still program structured if he desires. He may even load a package to ensure this. And if the code definition is long then the statement for high level would apply also. The code should be well scrutinized as to reason.

Note that using SP and ?CSP is a simple and effective way to catch many of the errors made in either case. They can even be used outside of the definitions of : and ; or CODE and ENDCODE, and never otherwise interfere. I am not saying there should be NO compiler security at all. If I had this viewpoint I would not have bothered to write the article. Even I feel much more comfortable with an application that I have programmed after it has been successfully loaded into a secure system. But I do object to having to program around the compiler security (which wastes time and introduces errors) when I wish to load a perfectly correct program which the security does not like. With an optional package, I can check my application as desired, but do not have to fight the compiler security to get the job done. Or, I can have the security package resident. I make the choice.

George W. Shaw<br>SHAW LABS LTD.<br>PO Box 303<br>San Lorenzo, CA 94580



FIG Convention Coming - Nov. 28

FORTH DIMENSIONS III/3

Here is a very short contribution, a compiler extension, which has been helpful to me. I want :s share it with all FIGgers; perhaps it can zecome a fig-FORTH standard.

Often in creating a definition, we want to -est or output an ASCII character, using words .. <e EMIT or = or possibly even in a CASE :satement. These are normally supplied as .teral numbers in the current radix. These zompile into the usual dictionary pair of LIT 'ollowed by the literal value. The difficulty is :hat we must either determine the ASCII code experimentally beforehand, or else reach for some reference list (usually in the wrong radix).

This compiler extension allows any editoresceptable character to be displayed in its real 'arm while compiling into a normal literal zair. While this may prove to be a minor help I: edit-time, the resultant source code is much -ore readable at a later time, and is self:2mmenting, both highly desired features of any : :ogramming language.

The new word is ASCII, and it is followed by I literal character. The definition of ASCII is $\therefore$ mple:

## : ASCII BL WORD HERE 1+C@ [COMPILE] LITERAL; IMMEDIATE

$\therefore$ is made immediate so that it executes during :s npile-time. WORD takes the next input-:-:-eam text, delimited by a blank, and places it $\equiv:$ HERE. Then the first character is placed on :e stack for use by LITERAL, which has been serced to be compiled into our definition. What zuld be easier?

Formerly, we had to write 65 EMIT to outDt the letter "A" (assuming decimal radix). - ow we can write ASCII A EMIT, clearly the :etter for everybody's understanding. The :mice of the word ASCII is open to change, but : $\boldsymbol{r e}$ idea is a valuable addition to our efficient .se of the language.

That's my contribution. I hope others can -se it to improve their work. Thank you for :-oviding a medium for ideas.

## MULTIPLE WHRLE SOLUTION

1 have no way of knowing whether this solution to the multiple WHILE problem is generally known, though I am sure that many people must be using it. The note has been kept as short as possible, and could easily be expanded.
(: ENDWHILE 2-?COMP 2 ?PAIRS could be simplified to : ENDWHILE ?COMP 4 ?PAIRS probably-it weakens the ENDIF analogy a little.

Many of your readers may not be familiar with ENDWHILE as a means of achieving multiple WHILEs in a BEGIN loop. It is simple and convenient, but not elegant. ENDWHILE is used in the construction

BEGIN. . .(test) WHILE. . .(test) WHLLE. . . ENDWHILE ENDWHILE AGAIN or

BEGIN. . .(test) WHILE. . .(test) ENDWHILE UNTIL

with one ENDWHILE for each WHILE in the loop.

The definition is

## : ENDWHILE 2 - ?COMP 2 ?PAIRS HERE <br> 4 + OVER - SWAP ! ; IMMEDIATE.

It causes WHILE to compile a branch to the word following AGAIN or UNTIL, and is directly analogous to ENDIF (THEN). It can be easily understood by comparing the definitions of WHILE and IF, and ENDWHILE and THEN.

A similar ENDWHILE can be defined for use in the ASSEMBLER vocabulary.

The ENDWHILE construction is awkward (poor English) but simple, and is worth using until something better is decided on.

## Julian Hayden

2001 Roosevelt Avenue
Vancouver, WA 98660

Raymond Weisling
Jln. Citropuran No. 23
Solo, Jawa Tengah
Indonesia

# FORTH STANDARDS CORNER 

Robert L. Smith

The word WORD has caused implementers of the 79-Standard a certain degree of difficulty. The definition of WORD as it appears in the FORTH-79 Standard is as follows:

WORD char -- addr
Receive characters from the input stream until the non-zero delimiting character is encountered or until the input stream is exhausted, ignoring leading delimiters. The characters are stored as a packed string with the character count in the first position. The actual delimiter encountered (char or null) is stored at the end of the text but not included in the count. If the input stream was exhausted as word is called, then a zero length will result. The address of the beginning of this packed string is left on the stack.

There are a number of problems with the definition as it stands. Later I will suggest a slightly modified definition which should clarify the apparent intent of the Standards Team, although some of the problems will remain for the present.

1. The phrase "non-zero delimiting character" presumably means that char must not be the null character. An error condition should be specified if char is found to be zero.
2. The character count is to be atored in the first character position of a packed string. That could mean that the character count could not exceed 127. Since a string holds a sequence of 8 bit bytes, the Clarification Committee of the Rochester Standards Conference felt that the term "character position" was a typographical error that should have been "byte position", thus allowing a string count up to 255 characters.
3. Since the source string could be as long as a block (1024 bytes), the character count could exceed 255. This case should be specified as an error condi-
tion. The action to be taken on an error condition depends on the implementation. A number of schemes have been proposed, but there are none that are completely satisfactory. Many people, including this author, feel that any count should be allowed.
4. The definition uses the phrase "actual delimiter encountered (char or null)". I do not believe that the Standards Team meant to required implementations to use a null as a universal delimiter, although many undoubtedly will. The sequence in which the above-mentioned phrase appears probably means that if the end of the input stream is encountered before the specified terminating character is seen, then a null should be appended at the end of the packed string instead of the specified terminating character.
5. Note that in addition to being a terminating delimiter, char also specifies initial characters to be skipped. That property makes WORD very difficult to use in conjunction with strings which may have a zero length. An example of a zero length string is the null comment ( ). If one attempts to use WORD in a straightforward manner to enclose the command terminated by the right parenthesis, he will find that it and all succeeding text will be skipped! Since under the Standard, the use of WORD is about the only way that one has access to the contents of the text input buffer, this limitation appears to this writer to be unreasonable.

I believe that the following definition of WORD meets the essential intent of the Standards Team, and clarifies the problems stated in (1-4). in order to not add to the confusion, I have put a new serial number on the definition.
WORD char -- addr

Receive characters from the input stream according to the delimiter char and place the characters in a string beginning at
addr+1 . The character count is placed in the byte position at addr. An error condition results if char is an ASCII null or if the count exceeds 255. Initial occurrences of char in the input stream are ignored. If char appears in the input stream as a terminating character, it is appended to the string but not included in the count. If the input stream is exhausted before char is encountered as a terminating character, the terminating character null is appended instead of char. A zero length will result if :he input stream is exhausted when WORD is called.

The problem of the character count limita$\therefore$ on could be considered in the future. One : mole approach would be to use a full word for - e character count. Another would be to elimate the character count and always append a - 11 at the end. The user could then do his own :氵anning. The problem of null length strings -ould be "defined" away by making null length $\because r$ ings illegal. I think that that is a poor :Lution. The real problem is that WORD is : orly factored. As usual in FORTH, the less a ~ord does, the more useful it becomes. The :-ocess of scanning for initial delimiters should $\therefore$ separated from the process of scanning for :erminating delimiters.

## NEW PRODUCT

Z8002 Software Development System under IP/M ${ }^{\text {® }}$ or Cromemco CDOS. Includes cross zompiler and a number of utilities. Available on 5 th $^{\prime \prime}$ or $8^{\prime \prime}$ disks. Price: $\$ 4,000.00$

Inner Access Corp. 517K Marine View Belmont, CA 94002 (415) 591-8295

## "Starting FORTH" Available NOW!

\$16.00 Paperback
\$20.00 Hardcover
Mountain View Press
PO Box 4656
Mt. View, CA 94040
(415) 961-4103

## THE FORTH-79 STANDARD-A TOOL BOX?

George W. Shaw, II

As a vendor of a version of the FORTH language, and a self-proclaimed priest of the FORTH religion (I carry a soap box in my back pocket and will mount upon it at a moments notice), I am very interested in the best standardization of the FORTH language possible. There are many items in the ' 79 standard which need work. Many cannot, and maybe should not, be changed this time around, but will have to wait for subsequent standardization efforts. To this end, I am conducting interviews to compile as complete a list of problem areas and solutions as possible. I would like to thank all of the people who have spent time giving me the input, comments and ideas which are the inspiration for this article.

Much discussion centers on the defining: "What is a ' 79 standard program?" Many of the questions are similar to "Can $I \times x x$, and will it be standard?", or "My system has a zzz which does more than the standard says. Is it standard?". These are the wrong questions. Granted, many of these questions could be answered by more explanatory text within the standard. But, in general, the real question is "What does a stendard mean?", or better "What is the FORTH-79 Standard?'

The ' 79 standard very clearly defines itself. But, unfortunately, it seems that many people skip reading the first page of the standard and branch right into the glossary. If one is to read the first page, one notices a section of great importance:

## 1. PURPOSE

The purpose of this FORTH standard is to allow transportability of standard FORTH programs in source form among standard FORTH systems. A standard program shall execute equivalently on all standard FORTH systems.

This section very clearly states the standards purpose is "... to allow transportability of standard FORTH programs in source form..." Further, that the program "...shall execute equivalently..." The section previous to the above clarifies the extent:

## 0. FORWARD

The computer language FORTH was created by Mr. Charles Moore, as an extensible, multi-level environment containing the elements of an operating system, a machine monitor, and facilities for program development and testing.

States clearly of FORTH as "...containing elements of ..." the various environments. Thus, the standardized FORTH should be a language which contains only the elements necessary to allow the transportation and equivalent execution of programs between FORTH systems. This is even further limited by the definition of transportability.

## transportability

This term indicates that equivalent execution results when a program is executed on other than the system on which it was created. See 'equivalent execution'.

Which implies that a 79 standard system (in this case, a system which contains only the standard words) does not necessarily allow program creation (development). This is not to say that one could not define within the standard the additional tools necessary to develop programs. Only that the set of standard words may not be sufficient for development. (The additional words necessary for development is definitely an area to be looked at for the next standard.)

Considering the above definitions, I propose this answer to the title question of this article: The FORTH-79 Standard is to be a basic tool box upon which other devices can be built. From the definitions within the standard one should be able to build almost any other needed tool or application. We do not know yet if this is the case. It is extremely unlikely that the initial effort would have encompassed all design possibilities. The ' 79 standard is a first effort--a place to start from; a base from which we may begin to determine the minimum additions necessary to allow all tools or applications to be built transportably.

Yet, even with this understanding, it may be felt that the standard is incomplete. In a few cases this may definitely be true. A good example of this is in the text dealing with the
vocabulary mechanism. The standard seems extremely limiting and impossible to deal with. But, the solution is simple. Do as you have always done in FORTH. If a structure is inadequate for an application, define a structure which is adequate. The standard itself, by content, forces development in those areas which have not yet been fully developed. It forces new ideas, better solutions, and, hopefully, a better standard next time around by its own proper usage.

As for the two most asked questions mentioned earlier, read the standard carefully. Does it specifically or implicitly prohibit xxxing? If not, try to transport it to other systems. If you are still unsure, send the question to FIG, we'll work out a clarification and recommend it to the standards team. What if your systern does more than zzz says? Can it be made to do only what zzz says by possibly not exercising options? If so, it is probably standard. Still not clear? Send in the questions. We need them to make a better FORTH-79 Standard document.

There are areas of the tool box which may be cluttered by parameter testing or unnecessary words. Some areas may require better factoring. Much work has yet to do done. These areas need to be exposed. Write FIG about them. All input is greatly appreciated. I have found that each person sees different valid problems. Many are seen by all, but most people usually see at least one that has not been seen before; an application or solution which had not been considered.

When considering the FORTH-79 Standard, treat it as a basic tool box. Additional tools are applications from the point of view of the standard. Extend it as necessary. Can you add what you need by defining it only in terms of standard words? If not, what is the minimum necessary to allow you to do that. More definitions or more explanations? Experience is all that will tell. Send in your results.

George W. Shaw, II<br>Shaw Laboratories, Ltd.<br>P. O. Box 303<br>San Lorenzo, CA 94580

Book Review:

title: Starting FORTH<br>Author: Leo Brodie FORTH, Inc.<br>Copyright: 1981<br>Publisher: Prentice-Hall, Inc.<br>Price: $\$ 19.95$ (hardbound)<br>$\$ 15.95$ (softback)<br>Availability: Mid September, 1981<br>Review by: George W. Shaw II<br>Shaw Laboratories, Ltd.

In most books the useful information begins . chapter one, or later. Starting FORTH is an exception. Useful information starts in the = orward section of the book.

The book is designed to be interactive. dfter only two pages of chapter one, you are :yping at the terminal. It is seldom that a sentence will leave you thinking, "Now, what soes that mean?" Analogies are used throug--out. Any "buzz" words, or differences between ivstems, or phrases which might cause conusion are footnoted to explain in more detail. this presents extremely basic or nonessential . Jeas outside of the main text, allowing the 200k to be simple enough for the beginner, but - $כ$ t to become tiresome to the more knowledgeable. For example, in the sections of the book sealing with math, separate sections or foot--otes are presented to explain what integers are $\simeq$ what an absolute value is (for beginners). Or $\because$ give additional information about a faster a!gorithm than was used in an example in the nain text (for experts). Where appropriate, juizzes or exercises are interspersed within the shapters to help with understanding the naterial presented.

The book is written for the current "close: $0-79-S T A N D A R D "$ version of polyFORTH with mations or footnotes to indicate and explain :he differences from the standard. Throughout :he book, tables and lists are used to summarize and clarify the information presented. The aecasional tables of new words (in glossary 'orm) are of great help. They prevent having to ig through the text for the words to perform :he practice problems. At the end of each zhapter is a complete glossary of the new words. Also, at the end of each chapter are oroblems, with the answers in the appedicies.

There are even a few surprise questions to lighten the air.

Moving from the general to the specific, the value of this work becomes even more apparent in the following chapter by chapter review.

The Introduction is not just one introduction, but two: one for beginners (to computers) and one for professionals. The beginnner's section explains conceptually what computers and computer languages are, using an analogy (as will often be found) to simplify. the professional's section answers the usual skeptical questions of "What is" and "Where is" Forth with an impressive list of facts about the language and applications in which it has been used.

Chapter one, "Fundamental Forth", presents the basic concepts of dictionary extensibility and problem definition immediately, so that after only two pages, you are typing at a terminal executing commands and defining words. The text steps the reader through the complete development of a program and then illustrates its execution with the previously mentioned cast. The operation of the stack is then illustrated, and the format of glossary entries explained.

Chapter two, "How to Get Results", presents the basic four arithmitic operations, calculator and definition style, with conversions between infix and postfix notations. Practice problems and stack pictures are provided to ensure comprehension. The next half of the chapter covers the basic single and double precision stack operations with excellent stack pictures and quizzes to help along the way.

Chapter three, the "Editor (and Staff)", again looks at the dictionary, but in terms of redefinition and FORGETing of words. Forth's use of the disk is also described, along with LSTing, LOADing, and the word "(" for comments.

Chapter four, "Decisions, Decisions, . . .", illustrates the IF ELSE THEN structure of Forth; the various conditional tests, their uses and alternatives; and flags and how to manipulate them.

Chapter five, "The Philosophy of Fixed Point", expands upon the basic four arithmetic operations with some of the composite ( $1+, 2+$, etc.) and some miscellaneous operations. The operators for the return stack are introduced with examples of their use in ordering parameters for formula calculations. A discussion of benefits of floating or fixed point math is followed with instruction about scaling in fixed point to eliminate the need for floating point. Discussed slso are the use of 32 bit intermediate operators and the use of rational approximations in fixed point.

Chapter six, "Throw It For a Loop", discusses the operation of the various types of loops in Forth. A new cast of characters illustrate the "how" of DO LOOPs, nesting loops, using IF ELSE THEN inside loops, etc. BEGIN UNTIL and BEGIN WHILE REPEAT are also introduced.

Chapter seven, "A Number of Kinds of Numbers", is divided into two sections: for beginners and for everyone. The beginners section gives an excellent tutorial introducing the novice to computer numbers. This section describes in detail both signed and unsigned single and double length numbers. Also covered are arithmetic shifts, bit-wise operations, number bases and ASCII character representation. The section for everyone explains Forth's handling of signed and unsigned single and double length numbers for input, formatted output and mathematical operations. The effect of BASE on I/O, some usage hints, and mixed operations are discussed.

Chapter eight, "Variables, Constants, and Arrays", discusses the uses and operation of these structures. Both single and double length structures are introduced. Example problems are used to show various designs for byte and single !ength arrays. Factoring definitions is also discussed.

Chapter nine, "Under the Hood", presents a very clear, detailed, explanation of the various types of execution and structures within a Forth system. Of the many things examined are: text interpretation, ticking ('), compiling, vectored execution, dictionary structure, colon definition execution, vocabularies, the Forth memory map and its pieces. Much of the detail applies to
polyFORTH, but the theory is sufficiently general to apply to the operation and structure of most Forth systems.

Chapter ten, "I/O and You", discusses string and text manipulation as they relate to disk and terminal I/O. Block buffer and terminal buffe: access is discussed with notes for multi-user systems. String operators and string to number conversion are also covered.

Chapter eleven, "Extending the Compiler: Defining Words and Compiling Words", weans the reader from the friendly cast of characters as it shows the code behind the faces. All of the aspects of Forth compiler are discussed including: time periods, the various compilers inside Forth, DOES $>$ words and inmmediate words. D-charts are introduced.

Chapter twelve, : "Three Examples", presents three programming problems and their solutions as an example of good Forth style. Text manipulation is presented with a random paper generator; Data manipulation with a file system; and fixed point number manipulation with a math problem which would seem to need floating point.

Following chapter twelve are four appendices which contain the answers to the problems, the features of polyFORTH not discussed in the text, the differences from the '79 Standard and a summary index of the Forth words presented in Starting Forth.

On the whole, Starting Forth is very well organized and presented. On occasion a few topics seemed to appear out of nowhere, as the section on Factoring Definitions in the chapter about variables, constants, and arrays. But, these digressions only serve as short breaks from the subject at hand and do not detract from the organization of the material. The text is very complete and easily understood. I rate the book very highly for both the novice and intermediate Forth programmer.

# THE FORTH ENGINE 

## David Winkel

What can computer architects do to make their lives interesting?

It has been clear for some time that building =onventional Von Neumann computers is useful out dull. This in spite of large vendors' advertising literture which breathlessly announces new architectural advances for their .atest machines. Meyers' book has an entertaining discussion of the history of these "new" advances. For example, virtual storage goes back to the Atlas system (U. Manchester, :959).

How can we improve performance? It appears that there are two practical ways:
a. Engineering - faster components, pipelines, caches, etc., applied to conventional architectures.
b. Architectural - building fundamentally different computers.

The engineering approach has been remarkably successful as shown by Seymour Cray's products. These machines do an excellent job with Fortran, but conceal gaps that programners have adjusted to and, in fact, accept as theologica! necessities. For example, the array is a fundamental concept of Fortran, yet is only indirectly supported in hardware. Subscripts joing out of range is a common run time error out the hardware happily goes on with the wrong data pointed to by a bad subscript.

The architectural approach would reverse the procedure. Build hardware to support a anguage. We can do this at several levels, the owest being language-directed design where רardware features are added to support specific language features. An example would be Burroughs' concept of data descriptors to provide run time checking of subscript ranges. Another example would be a P-code machine. D-code is language-directed since it was proposed as an ideal machine for compiled Pascal. It would be less suited for FORTRAN for example. The general idea in languagedirected design is to mirror important highlevel language concepts in hardware. Semantic Gap is defined as the degree to which language
features are not mirrored in hardware. Thus, the semantic gap for ALGOL running on a Burroughs B6500 would be small, for PL/l running on CDC machines quite large.

If we reduce the semantic gap to zero, we have a direct execution machine where hardware mirrors all the constructs (both data and control) of the language. Good discussions and bibliographies are given in references 1 and 2.

Now we have the maximum in speed and the minimum of generality. The computer now runs only one language. What that language should be is a central question. The SYMBOL computer was an early, truly heroic, system built by Fairchild, to directly execute the Symbol language. ${ }^{3}$ This is a PL/l-like language with a great deal of power. System performance was spectacular and yet the entire exercise cannot be considered successful. A large part was due to language complexity which translated into hardware complexity. It was difficult to fix bugs and impossible to add features inadvertently left out.

What we need is a well-tested, simple language before we build a corresponding direct execution machine. FORTH is the obvious choice.

The goal of this research is to build the world's fastest FORTH engine. This is a nocompromise effort to force the hardware to mirror the language. We did not start by saying it must be built with bit slices, or PLA's, or ... In fact, an early paper design was done with bit slices and discarded because it was too slow.

The measure of speed is clock cycles per instruction. Clock rate, in turn, is a function of technology, not architecture. The machine currently runs at 333 ns but could be easily speeded up by using ECL or Schottky logic and faster memories.

The design cycle for a FORTH primitive proceeds as follows:
a. Pick a primitive such as DO or LOOP.
b. Postulate hardware data paths, stacks, registers, etc., that implement DO and mating primitives such as LOOP and +LOOP.
c. Make sure this hardware supports hidden logical concepts-in this case, I , $\mathrm{J}, \mathrm{K}$-and violates no other FORTH concepts.
d. Count clock cycles.
e. Repeat b-d until you can think of no more speedup possibilities.
f. Make an engineering choice for implementation. DO takes 2 clock cycles without overlap, 1 with. LOOP and + LOOP take 1 clock cycle. For the first machine, we use a 2 -cycle DO and reserve the l-cycle version for later enchancements. As a byproduct of this implementation we can support loop nesting to a depth of 1024 .

This process is repeated for each FORTH primitive. Finally, this collection of individually optimized hardware must be forged into a coherent whole that makes engineering sense. The result ${ }^{4}$ is not too surprising. There are data and return stacks plus separate stacks for loop control. Of course, the loop stacks are invisible to the programmer. An arithmetic unit operates from the data stack, etc. What is surprising is the mass of data paths required to support parallel operations such as 2SWAP in one clock cycle. The results are impressive. For the fig-FORTH primitives all but 4 can be executed in one or two clock cycles with the exception of multiply and divide which take 1 clock cycle per bit. The machine currently has $16 \mathrm{k} \times 16$ main memory with $1 \mathrm{k} \times 16$ stacks both extendable by 4 X . I/O is done with a slave 6809 , rith programmed access to the data stack and DMA access to main memory. Control is microprogrammed with a 2910 driving a $1 \mathrm{k} \times 60$ bit writegble control store. This follows Logic Engine philosophy so the user has very pleasant access to the micromemory for tailoring high-speed special purpose instructions.

Results for randomly chosen instructions are given below. All comparisone are based on a 1 MHz 6809 running fig-FORTH. The FORTH engine runs at 3 MHz .

| DUP | $99 \times$ faster | SWAP | $132 \times$ faster |
| :---: | :---: | :---: | :---: |
| [ | $101 \times$ faster | U* | $96 \times$ faster |
| ! | $114 \times$ faster | ROT | $624 \times$ faster |
| AND | $126 \times$ faster | DO...LOOP | $110 \times$ faster |

As a rule of thumb the speedup is a factor of 100. Why the 6809 (or any other computer) is so slow is an interesting question and will be treated in a more formal paper.

We have received a number of inquiries about machine availability. Does anyone really need a machine this fast? It is obviously a large (200+ IC) machine in the minicomputer class and will cost more than a 280 . I would appreciate hearing from readers about this as well as memory and $1 / 0$ requirements.

David Winkel 2625 Solar Drive \#5
Salt Lake City, UT 84117

## REFERENCES

${ }^{1}$ Advances in Computer Architecture, Glenford J. Meyers, Wiley, 1978.
${ }^{2}$ Southcon Conference, Atlanta, Georgia, 1981, Session 20/2, Phillip Crews.
${ }^{3}$ SYMBOL - A Major Departure from Classic Software Dominated Von Neumann Computing Systems. Proc. 1971 Spring Joint Computer Conf., AFIPS, 1971, pp. 575-587.
${ }^{4}$ Southcon Conference, Atlanta, Georgia, 1981, Session 20/4, David Winkel
${ }^{5}$ The Art of Digital Design, D. Winkel \& F. Prosser, Prentice Hall, 1981.

THE FORTH, INC. LNE EDITOR<br>S. H. Daniel<br>System Development Corporation<br>500 Macara Avenue<br>Sunnyvale, CA 94086

The upcoming publication of Starting =JRTH, which is destined to become the "bible" :: FORTH neophytes everywhere, provides an soportunity to upgrade the existing fig-FORTH - e editor at a very small cost in time and effort.

There are at least two good reasons why this -ograde should be done. The first is standardzation. A user of any version of fig-FORTH - :l. be able to step up to a polyFORTH system and use the line editor. Conversely, FORTH, $\therefore$ customers who try fig-FORTH will not have $\because$ !earn to use a different editor.

The second reason for adopting the $=:=\mathrm{yFORTH}$ editor is its increased flexibility $\equiv x$ ease of use. The current fig line editor uses -r.y the PAD for storage of user inputs for searches, deletions, and replacements. The ZIyFORTH editor employs both a FIND buffer $\equiv d$ an INSERT buffer, in addition to the PAD. - is allows both of the extra buffers to be : aded, and the contents reused several times, $-:$ :hout extra typing by the user. This makes :: mmands like D (Delete) and R (Replace) تsjecially useful.

3y taking a few hints from Starting FORTH, $\equiv \chi$ combining them with the existing editor, I - is able to write a line editor which is - - ctionally identical to the polyFORTH editor, $\because:$ which is in the public domain and can be sed by anyone.

## E'STEM REQUIREMENTS

This editor should run on any fig-FORTH : stem, including FORTH-79 Standard systems : the changes mentioned in the section = JRTH-79 Standard are made). The compiled e editor requires approximately 2 K bytes of -e nory, plus room in the system for the PAD E-d the FIND and INSERT buffers. It operates -.thin the confines of the default data and ?turn stacks.

A high level version of the word MATCH, used by the line editor for searches, is included for those who do not already have a version written in assembly language. If you intend to use this version of MATCH, screens 216 and 217 should be loaded prior to loading the rest of the line editor. Credit for this version of MATCH goes to Peter Midnight of Hayward.

## THE EDITOR COMMANDS

The word 'text' following a command indicates that any text typed after the command will be copied to the text buffer used by that command. The buffer contents will then be used when the command executes. If no text is typed by the user, the contents of that buffer (left over from the previous command or commands) will be used without modification in the execution of the command.

$$
\times \quad \text { eXtract } \quad(-)
$$

Copies the current line into the INSERT buffer, and removes it from the screen. All following lines are moved up, and line 15 is left blank.

$$
T \quad \text { Type } \quad(n-)
$$

Type line $n$ from the current screen. Set the cursor to the start of the line.
L List ( -- )

Like the FORTH word LIST, except that the current screen number is obtained from the variable SCR, rather than being typed in by the user.

$$
N \quad \text { Next } \quad(--)
$$

Increments the current screen number by one. This command is used just before the $L$ command, to allow the user to list the next sequential screen.

$$
\text { B Back } \quad \text { (-) }
$$

Decrements the current screen number by one. This command is also used before the $L$ command, to allow listing of the previous sequential screen.

$$
\begin{aligned}
& P \text { text } \quad \text { Put } \quad(-)
\end{aligned}
$$

Any following text will be copied into the INSERT buffer. The INSERT buffer will be copied into the current line, replacing its previous contents. If the text consists of one or more blanks, the current line will be erased.
WIPE Wipe ( -- )

Erases the current screen. Equivalent to the original CLEAR command, except that the user need not enter the screen number.
COPY Copy (from -2, to-1 - )

Copy one screen to another.

$$
F_{\text {text }} \text { Find } \quad(-)
$$

Any following text is copied to the INSERT buffer. The contents of the INSERT buffer are compared to the contents of the current line. If a match is found, the line is displayed with the cursor positioned immediately after the end of the string searched for. The F command, with no following text, is exactly the same as the previous editor command $N$. If no match is found, the requested string is echoed to the terminal and the error message "NONE" is output.

$$
\text { E Erase }(-)
$$

Erases backwards from the cursor, according to the number of characters in the FIND buffer. This command should only be used immediately after the $F$ command.

$$
\begin{aligned}
& \text { D Delete }(--) \\
& \text { text }^{\prime}
\end{aligned}
$$

Any following text is copied into the FIND buffer. The D command is a combination of the $F$ and $E$ commands. The string in the FIND buffer is matched against the contents of the current line, and if a match is found, the found string is deleted from the line.

$$
\begin{aligned}
& \text { TILL Till } \quad(-)) \\
& \text { TILL text }
\end{aligned}
$$

Any following text is copied into the FIND buffer. Starting from the current cursor
position, TILL searches for a match with the contents of the FIND buffer. If a match is found, TILL deletes all the text on the line from the current cursor position up to any including the end of the matched text.

$$
\begin{aligned}
& \text { S Search (last screen\#+1 --) } \\
& \text { Stext }
\end{aligned}
$$

Any following text is copied into the FIND buffer. Starting at the top of the current screen and continuing until the bottom of the screen immediately before the screen number on the top of the stack, 5 searches for a match to the contents of the FIND buffer. Whenever a match is found, the line containing the match will be typed out, along with the line number and screen number in which the match occurred. Because of the way FORTH handles loops, the number on the top of the stack must be one higher than the highest screen to be searched.

$$
\begin{aligned}
& \text { I text }
\end{aligned} \quad \text { Insert } \quad(-)
$$

Any following text will be copied into the INSERT buffer. The I command copies the contents of the INSERT buffer into the current line, starting at the current cursor position. Any text to the right of the cursor will be pushed to the right and will be pushed off the line and lost if the total length of the line exceeds 64 characters.

$$
U_{\text {text }} \text { Under } \quad(--)
$$

Any following text will be copied into the INSERT buffer. Spread the screen at the line immediately below the current line, leaving a blank line. All following lines are pushed down. Any text on line 15 will be lost. The contents of the INSERT buffer will be copied into the blank line, and that line will be made the current line.

## R Replace ( -- ) <br> $\mathbf{R}$ text

Any following toxt is copied into the INSERT buffer. The $R$ command operates as a combination of the $E$ (Erase) and 1 (Insert) commands. Starting at the current cursor position, and working backwards towards the start of the line, text corresponding to the
.ength of the contents of the FIND buffer is seleted, and the contents of the INSERT buffer a:e inserted into the line. Since the contents of : e FIND buffer determine how much text will ze erased, the $R$ command should only be used nmediately following the $F$ (Find) command.
M Move (Block非, Line\# -- )

Zopies the current line into the INSERT buffer, : en copies the INSERT buffer to the block, sjecified by Blockझ, UNDER the line specified $=$ LINE\#. The original block number is :estored, and the next line in the block becomes :רe current line. This allows sequential lines to ye moved with a minimum of keystrokes. One j)fortunate side-effect of this command is that $: 3$ move something to line 0 of another screen, . כu must first move it UNDER line 0 , using the sommand $x x \times 0 \mathrm{M}$, make screen xxx current, and then extract the old line 0 , moving everything else up.

$$
\uparrow \quad(--)
$$

Jsed as a terminator for all commands allowing :ext input, such as $P, F, R$, etc. Allows more :han one command to be entered on a single .. ทe, e.g.,

## $3 T P$ This is line $3 \uparrow L$ (cr)

- l though useful, this feature does preclude the use of the " " as a character in any text to je put on a screen.


## ILOSSARY

The following glossary addresses all the = JRTH words in the line editor except the estual editing commands, which are discussed zoove.
${ }^{\text {EXT }}$ (delimiter -- )
Any following text in the input stream, up to but not including the delimiter, is moved to the PAD. The length of the input string is stored at PAD, with the actual string starting at PAD+1. In FORTH-79 Standard systems, if no text follows in the input stream, a length byte of 0 will be stored. In non-Standard systems, a
length byte of 1 will be stored, but PAD +1 will contain a null to indicate the absence of text.
(LINE) (Line\#, Screen\# -- Buffer Address,64)
Using the line and screen numbers, computes the starting memory address of the line in the disk buffer. May not be necessary in FORTH-79 Standard systems, depending upon implementation. Should already be present in earlier implementations.

LINE (Line\# -- Buffer Address)
Ensures that the line number is within the legal range of the current screen, then uses (LINE) to set the starting address of the line in the disk buffer.

## WHERE (Offset, Block\# --)

Used when a compile time error occurs during loading. Converts the block number to a screen number, makes that screen current, and prints the line in which the load error occurred. Underneath the line in error, the cursor is printed to show the approximate location of the error. Enables the EDITOR vocabulary as it exists. Strictly speaking, this is not part of the polyFORTH editor, but it is a highly useful tool.

## \#LOCATE(--Cursor offset, line\#)

Uses the current cursor position to compute the line number which contains the cursor and the offset from the beginning of the line to the current cursor position.
\#LEAD (--Line address, offset to cursor)
Computes the beginning address of the current line in the disk buffer, and the offset from the start of the line to the current cursor position.
*LAG ( -- cursor address, count after cursor)

Computes the address of the cursor in the disk buffer and the count of characters remaining on the line after the cursor.
-MOVE (from address, to line\# --)
Moves a line within the disk buffer to the line specified, completely replacing the previous contents of that line.
?MOVE (destination buffer address -- )
If any text has been entered into the PAD by TEXT, moves that text to the specified buffer. Used to load the FIND and INSERT buffers for searches, etc. If no text was in the PAD, no action is taken.
>LINE\# (- current line number )
Uses the stored cursor location to compute the current line number.

FIND-BUF ( - )
Establishes the FIND buffer a fixed distance above the current address of the PAD.

INSERT-BUF ( - )
Establishes the INSERT buffer a fixed distance above the FIND buffer.
(HOLD) (Line\# -- )
Non-destructively copies the contents of the current line to the INSERT buffer.
(KILL) (Line: - )
Replaces the specified line with a blank line.
(SPREAD) ( - )
Spreads the screen, starting at the
current line, pushing all lines below the current line down, and leaving the current line blank. Any text on line 15 is pushed off the screen and is lost.

DISPLAY-CURSOR ( -- )
Displays the current line with the cursor in place.
(R) ( -- )

Replaces the current line with the contents of the INSERT buffer. Used as the primitive operation for the $P$ command.
(TOP) ( -- )
Resets the stored cursor position to the top of the screen.

ILINE ( -- Flag)
Scans the current line for an exact match with the contents of the FIND buffer. If a match is found, the stored cursor position is updated.
(SEEK) ( -- )
Starting at the current cursor position, searches the rest of the current screen for an exact match to the contents of the FIND buffer. If no match is found, the contents of the FIND buffer are typed and the error message "NONE" is output.

## (DELETE) (Count -- )

Starting at the current cursor position, text is deleted backwards (towards the beginning of the line), according to the count. The remaining text on the line is moved to the left and the end of the line is filled with blanks.
(-)
Copies any following text to the FIND buffer and searches the
current screen for a match. Uned as the primitive operation for the $F$ and $D$ commands.
E) (-)

Uses the length of the contents of the FIND buffer to set the count for (DELETE). Used as the primitive for the $E$ and $R$ commands.

A variable used by the $S$ command to count the number of lines output to the screen and printer.

3UMP ( - )
Increments the number of lines output and sends a page eject when 56 lines have been output. Used by the 5 command to handle pagination on the console and printer.
FORTH-79 STANDARD
The following changes should be made to the .ad screens shown in order to bring the line editor into conformance with the FORTH-79 Standard:

Screen
209

Line(s)Change
9,10 The FORTH word $R$ should be changed to R@.

214

212
5,6,7 The FORTH word R should be changed to R@.

3 The 0 preceding the word VARIABLE should be deleted, since variables are initialized to 0 automatically under the FORTH-79 Standard.

12 The word 1+ may be deleted, since the FORTH79 Standard specifies that $a$ length byte of 0 will be stored when WORD finds no text in the input stream.

## ERROR MESSAGES

Only two error messages are output by the line editor:

NONE Indicates that no match was found on the current screen corresponding to the contents of the FIND buffer.

## NOT ON CURRENT EDITING SCREEN

Indicates that the line number passed to the word LINE was outside the legal range of $0-15$ decimal.

```
S0R 200
```



```
FORTH DEFINITICNS HEX:
    TEXT & eceet rallowina lext to FAL;
    HEEE C/L 1+ ELFWKS NOFS,
    : LINE GFFFG fun < relative to SCR, lewve oddrest or lime,
        OUP EFFFG FAND
        IF ." HOT OH CURRENT EOITING SGREEN" QUIT THEN
        SCR <LINE> DROF:
    ->
```

NOCPBULFRY' EEITOR IMTEUIRTE NEX
: HHERE \&mint sereen % and imbue of orrer )
DLF BJSCR / WUF SCR !."SCK " DECIMFL

```

```

                SWNP CRLMOO CRL * FUT ELCNU
    EDITOR DEFINITIONS
MOCATE C/ M MCN"; cursor offset-2, line-1;
一>
202
MERD, \#LFG, MOCLE, BUF-HMUE
Eldioir sivi.,

```

```

        LMG < cursor adr-2, ccunt after aursor-1>
    ```

```

    MOUE <M { mose rrom nom-2, to linm-1 - >
        LINE CLL CMOUE LPDATE :
    : EuF-MOUE (move text to buffor-1, if ams-)
        PACH 1+ C& SWF CML 1+ CMMUE
        ELSE DROF CL 1t CNUNE
        THETH : ->
    SCR * 205
AINES, FINDMEUF, INSERT-GUF
\&IME. ( corwert current eureor mosition to binm)
\#LGCATE SUCP DFOP %

```

```

        PFOD 50 + %
    INSERT-EUF <O <Eurfor used for all imertions`
    ->
    SCR \204
(HOLD-, (KILL-. (SPREAD-. X
8197e7 sht ,

```

```

<KILL) CA munes erase line-i with blarks)
LINE CL BLEWGS LPDFTE :
(SPRCEFD) (suread, making lirew blark)
\INEW OUP 1 - DE
0O I LINE 1 1* HOUE -1 HOOF (KILL) :
: X \& delete linee from bloak. Dut in insert eutfor)
MINE OUP (HOLD) OF DUP ROT
->
SCR * 205
C DISFLAM-CIPSCR, T, L
GL6713 SN4,
DISPLAM-GUSC
CR SPFCE OLERD TVPE SE EMIT
MLRG TYPE GLOCRTE DRCP%
T (C, <tue linat-1 )
CA * RN ! O DISPLRN-ONSOR \&
L SCR LIST ; \ list cumreme serwen )
->

```

```

ScR 210
<210
810715 510 )
(F) ( find occurence of followina text )
FE TEXT BN-EUF BUFMOUE
SEEK) :
(F ( find and dimplay followine text )
(F) DISPLAN-CURSOR:
((E) ( mrase beckwards from cursor)
FIND-EUF CO (DELETE)
E (crase and dimplam line)
(E) DISPLRN-CURSOR3

```

SCR 212
\(8<C O U\)
212
COUNTER, BUNP
819707 gnd
UPRIPELE COLNTER
    Eurp ( the line number and hande omine)
    1 COUNTER +!
                        COMTER
        \(38>\) IF COUNTER:
        CR CR OF MESSFAE OC EMIT THINH:
    \(\rightarrow\)
    \({ }_{5}^{213}\)
    \& 5
                                (Trom currant to screen-1 for strine)
    QC EMIT SE TEXT COUNTER :
    FIND-AE OEFHCUE
    SCR DUP \(冫 R\) DO I SCR :
    (TOP)
    EEGIM
        ILINE IF DISPLOV-CURSOR SCR ? EUTP TMEN
        JFF R* <
    UNTIL
    LOOP RD SCR : :
\(\rightarrow\)
(insert text watrin lins)

SE TEXT INSERT-gUF EUF-WUE ( if ans) IMSERT-EAF COUNT OLIG ROT OUER MIN PR
 DUP MERE R CMOVE S TrCin old curear to MESE MSFE MESD + R> CMOVE (HEFE Lo equmon locidion) \(R\) CMAVE UPNHTE (PFU LO odd eursor) OISPLFY-CREGK ( look at now line)
    CL RW +! (SPRERU)P \&
    \(\rightarrow\)
    - 215
    -R. \(M\)
                                    918715 SHD
    : \(R\) (replace found text with imsert burfor)
        (E) 12
    M
                            (moue from curpent dine on curpert screen)
    \{ to scromr 2, UNER line 1 )

        R MR (HOLD) ( mou current limet io inemt burfers)
        AIME (HOLD)
        Supe SCR ! ( eat new screen )

        \(1+C R\) ( CH
        (SFREFD) (R)
            (store imeert buffier in new sereen)
        \(R\) R \(\mathrm{CL}+\mathrm{R}\) ! (tet orisimal oursor co next line)
        R) SCR ! \(:\) ( Pestore onistimal screm )
    FORTH DEFINITIONS DECIMFL


\title{
RECURSION AND THE ACKERMANN FUNCTION
}

\author{
Joel V. Petersen
}

Recursion involves the calling of a program by itself. An example of where recursion might be used is in the parenthesis handler of an algebraic string parser. Every time the parser encounters a left parenthesis, it calls itself; every time the parser encounters a right parenthesis, it completes a call of itself. Recursion is somewhat difficult to explain and very difficult to use properly. However, the implementation of recursion in any language can be tested with a program called the Ackermann Function. This is a recursive function of two variables which is almost impossible to explain. The following is an implementation of the function in PASCAL.

VAR K,J: INTEGER; CALLCNT;INTEGER;
FUNCTION F(K,J: INTEGER): INTEGER; BEGIN

CALLCNT \(:=\) CALLCNT +1 ;
IF \(K=O\) THEN
F: \(=\mathrm{J}+1\)
ELSE
IF \(J=O\) THEN
\(F:=F(K-1,1)\)
ELSE
\(F:=F(K-1, F(K, J-1)) ;\)
END(*ACKERMANN FUNCTION*);
Recursive programming as illustrated in the PASCAL example is not possible in FORTH. A program can not invoke itself simply by using its own name while defining that word. However, recursion is not difficult at all to achieve:
(FIG-FORTH)
: MYSELF LATEST PFA CFA , ; IMMEDIATE
(NIC-forth)
: MYSELF LAST @ @ \(2+\), ; IMMEDIATE
MYSELF simply places the address of the code field of the word being defined into its own definition. Thus, whenever the program needs to invoke itself, the word MYSELF should be used instead. The Ackermann Function now
becomes:
(FIG-FORTH)
O VARIABLE CALLCNT
: ACKERMANN (IJ-F)
1 CALLCNT +!
\(\mathrm{O}=\mathrm{IF}\)
SWAP DROP \(1+\)
ELSE
DUP
\(\mathrm{O}=\mathrm{IF}\)
DROP 1-1 MYSELF
ROT ROT DROP 1- SWAP MYSELF
THEN
THEN;
(NIC-forth)
VARIABLE CALLCNT
: ACKERMANN (IJ-F)
1 CALLCNT +! OVER THEN

DUP
THEN
2DUP 1- MYSELF
-ROT DROP 1- SWAP MYSELF
ELSE
DROP 1-1 MYSELF
ENDIF
ELSE
SWAPDROP 1+
ENDIF ;
For comparison, the Ackermann Function was tested on the Nicolet 1280 20-bit processor in both (compiled) PASCAL and NIC-forth. The \(K=3, J=5\) function took 8 seconds in (compiled) PASCAL and 12 seconds in NIC-forth. (As an aside, the addition of a simple hardware mod to the 1280 processor to speed up NEXT in NICforth reduced this to 9 seconds! Who says inline coding is so much faster than indirect threaded code!)

When attempting to try the Ackermann Function, one must allocate lots of room for both the parameter stack and the return stack. Every time the function is called, there must be two elements on the parameter stack, thus the parameter stack will fill up approximately twice as fast as the return stack. The \(K=3, \mathrm{~J}=6\) function
requires over 1000 elements on the parameter stack and over 500 elements on the return stack at its deepest point. When the \(K=4, J=1\) function was tried, the program finally crashed after five hours with the return stack containing over 5000 elements!!

The results of the simpler Ackermann Functions are given below. \(F\) is the value returned by the function. CALLCNT is the count of how many times the program called itself. MAXDEPTH is the maximum depth attained by the return stack.
\begin{tabular}{|c|c|c|c|c|}
\hline K & J & F & CALLCNT & MAXDEPTH \\
\hline \(\overline{0}\) & \(\overline{0}\) & \(\bar{T}\) & & \\
\hline 0 & 1 & 2 & 1 & \\
\hline 1 & 0 & 2 & 2 & \\
\hline 1 & 1 & 3 & 4 & 3 \\
\hline 1 & 2 & 4 & 6 & \\
\hline 2 & 0 & 3 & 5 & \\
\hline 2 & 1 & 5 & 14 & \\
\hline 2 & 2 & 7 & 27 & 8 \\
\hline 2 & 3 & 9 & 44 & 10 \\
\hline 2 & 128 & 259 & 33669 & \\
\hline 3 & 0 & 5 & 15 & \\
\hline 3 & 1 & 13 & 106 & \\
\hline 3 & 2 & 29 & 541 & \\
\hline 3 & 3 & 61 & 2432 & 63 \\
\hline 3 & 4 & 125 & 10307 & 127 \\
\hline 3 & 5 & 253 & 42438 & 255 \\
\hline 3 & 6 & 509 & 172233 & 511 \\
\hline 4 & 0 & 13 & 107 & 16 \\
\hline 4 & 1 & ?? & ?? & ?? \\
\hline
\end{tabular}
\(\cdot\) - C-forth is the implementation of FORTH on -e NICOLET INSTRUMENT CORPORATION \(\therefore 30 / 1280\) series computers. This computer is a \(\because\)-bit minicomputer with a 19-bit address scace.)

> Joel V. Peterson
> Nicolet Instrument Corp.
> 5225 Verona Road
> Madison, Wisc. 53711
> ? 608\() 271-3333\)

三- - A great article, but watch out. Most \(\approx=O R T H\) implementations have insufficient ::ask space to execute this function. Programs sould be reviewed for compatibility.)

\title{
REVIEW \\ A Brief Review of the Manuals for the PET/CBM fullFORTH \(+\mathrm{V1.3/4}\)
}

\author{
by Jim Berkey
}

Complete system is available from IDPC Co., Box 11594, Philadelphia, PA 11916 for \(\$ 65\) (plus shipping?). Includes about 70 pages of documentation and a 54" diskette (not reviewed).

IDPC's fullFORTH+ is noted to have taken a person-year to be developed by an experienced programming staff. I give them a triple E for effort, but the product is, at best, rough.
fullFORTH + is described as "s complete implementation of the FORTH language, as defined by the FORTH Interest Group." If this is true of the disk, then there are glaring technical errors in the glossary, whose definitions deviate substantially from the FIG manual. One example from + LOOP : "If the counter and limit values are equal, either before or after adding the increment, the DO loop is exited . . ." If you take this literally, the counter (read "index") is compared to the limit twice-once before and once after the incre-ment--and exit can never occur on greaterthan, as it does in the FIG model.

On the plus side, the package includes 6502 assembler, screen editor ( \(n\) ( PETs ), printer support, and floating point routines. These are nice to have, but from the samples of use shown, I suspect the presence of endless small inconveniences. To be fair, endless small inconveniences are a built-in feature of CBM disk systems which fullFORTH + has not corrected.

I can't recommend fullFORTH+ for any but the desperate, because of two central problems: (1) the manual reveals a mangled view of the FIG model, and (2) fullFORTH + was probably not implemented originally for the PET/CBM.

\author{
FORTH, INC. NEWS
}

\section*{BETTER SUPPORT PROMISED THROUGH FORTH, INC. AND TECHNOLOGY INDUSTRIES MERGER}

FORTH, Inc. and Technology Industries, Inc. of Santa Clara, CA., have announced a merger. This means that FORTH, Inc. will become a wholly owned subsidiary of Technology, and the present shareholders of FORTH will become shareholders of Technology.

Technology Industries is a new company founded in February 1981 by John Peers. Peers is best known as founder and former chairman of Logical Machines Corp. of Sunnyvale, CA. This very successful company manufactures and sells business computers that feature a "programmerless" language called Adam, designed by Peers.
"The principle change that everyone will notice," said FORTH, Inc.'s president, Elizabeth Rather, "is that we'll be doing a lot more of what we do best--selling and supporting high quality professional FORTH systems and applications-and doing it even better. We're expanding our staff and investing heavily in equipment training."

FORTH, Inc. will operate with its individual identity, retaining the same name and operating structure. Technology Industries will be the "parent" of several other new companies as well. Each will specialize in hardware designed around and featuring FORTH. "Membership in this group will provide us with the opportunity to do some things I've wanted to do for years," said Chuck Moore. "Im extremely excited about these plans."

\section*{EXPAR ISION CONTINUES}

FORTH, Inc.'s growth in recent months has included two significant additions to management.

Joseph "Skip" Reymann, formerly with GOULD NAVCOM of El Monte, California, has joined FORTH, Inc. as vice president of operations. Reymann has extensive experience in both the technical and business aspects of
program management. He has degrees in physics, finance, and corporate and contract law.

Robert E. Smith, Jr. is FORTH, Inc.'s new vice president of sales and marketing. Smith has over ten years of experience marketing application software for minicomputers. He has already tripled the size of the marketing department and plans to triple it again within eighteen months.

Other important additions to the staff include two people in the accounting department and three sales and marketing representatives. The products department has been reorganized with Leo Brodie, author of Starting FORTH, acting as manager. The publications department has grown by two, and three general support staff members have come on board.

\section*{RECENT APPLICATIONS}

FORTH, Inc. recently signed a contract with International Business Services, Inc. in Washington, D.C., to supply hardware and software to the United States Forest Service.

FORTH, Inc. will provide the hardware and update and enhance the software for a highresolution map analyzer system. The system will work with digitized data from existing contour maps in rester format.

The raster-scanned maps will be displayed on a high resolution (1024 \(\times\) 1024) image system. A PDP-11/44 is then used to follow a given contour line and convert it to a string of vectors. Operator assistance is required in selecting a contour line, labeling, handling breaks in data, and making corrections from the original map. Operator input is via a track ball interface and alpha-numeric CRT.

Dick Liston of USFS has used FORTH for several years developing a prototype version of the system using miniFORTH on a PDP 11/05.

\section*{MARKETING COLUMN}
Q. I've written several programs that all my friends think are excellent; what is the best way to market them?--M.L., New Mexico
A. There is no universally "best" way to market anything, and that includes computer programs. Generally speaking, however, planning is your best ally. Since you have already received some feedback (and I assume you are certain that it is valid and not just your friends being politely supportive), it makes sense that persons that closely match the profile of your friends in terms of need, occupation, income, etc. would be your best prospects. Simply put, marketing under these circumstances will consist of finding a way to communicate effectively and cost effectively with this target group.
Q. I've run a number of ads for software I have developed and while I have sold some, I just don't seem to make any real money for the time I am putting in-what am I doing wrong?--R.B., Sandusky, Ohio
A. Your problem points up many areas that do not occur to the amateur entrepreneur. In the interests of brevity, I will touch on a few of the more significant as being instructive to our readers.
- PRODUCT--in this area you may be promoting a product that serves no real need or is competing with an already established vendor.
- PRICE--your price may be too high, causing your potential customers to seek other sources or do without; or, more commonly, your price may be too low, causing you to perform excessive labor in selling and servicing your accounts for the amount you are charging.
- MEDIA--you may be advertising or selling to the wrong audience. If you have failed to research your market and are running ads based on who's cheapest as opposed to who's reading (prospect profile), you are unlikely to achieve any realistic sales.

Remember your media should be purchased on the basis of cost per prospect, not cost per 1,000 .
- MESSAGE--you may be saying the right thing to the right people, but in the wrong way. Part of your test marketing should be to give your advertising and sales copy to a rank amateur and see if what they think you are saying is the same thing you think you are saying.

The above list is by no means all-inclusive, but these are the areas you should start looking into first.
Q. Is there any way of selling my programs other than by buying ads, etc.?--B.C., Walnut Creek, CA
A. Yes. One of the most common ways is to have your software merchandised through any number of firms that specialize in this field. Basically the way they operate is to contract with your for ownership of your software and pay you a royalty on sales-much like an author receives from a book publisher. Naturally, the royalty is nowhere near the amount you would receive if you sold your software directly to the consumer yourself; but considering that you have no risk and your time is free to develop additional products which in turn can be sold, the reduced percentage is still often the best way to go. The point is that it isn't how large a percentage you receive that is important--but how much money you make.

Questions of general interest regarding the marketing of software will be answered in each edition in this column. Because of time limitations, it will not be possible to provide private answers either by phone or mail. In the interests of personal privacy, questioners will be identified by initials only. Questions should be addressed to:

\author{
MARKETING COLUMN Editor, FORTH DIMENSIONS \\ PO Box 1105 \\ San Carlos, CA 94070
}

\section*{HELP WANTED}

\section*{FORTH PROGRAMMERS}

Openings at All Levels At FORTH, Inc.

Programmers experienced with mini/micro computers and peripherals to produce new polyFORTH systems and scientiflc/industrial applications. Degree in science or engineering and knowledge of FORTH essential.

\section*{PRODUCT SUPPORT PROGRAMMER}

DUTIES: Responsible for maintaining existing list of software products, including polyFORTH Operating System and Programming Language, file management options, math options and utilities and their documentation; and providing technical support to customers.

Requirements include:
Good familiarity with FORTH-preferably through one complete target-complled application. Assembler level familiarity preferred with the 8080, PDP/LSI-11, 8086, M6800, CDP1802, NOVA, IBM Series I, T1990. Communication skills are essential.

\section*{PROJECT MANAGER}

Project manager to supervise applications and special systems programming projects: writing proposals, setting technical specifications, customer liaison, hands-on programming, and supervision of senior programmers.

\section*{SENTOR INSTRUCTOR}

Experienced in course writing and development, technical education in computer software, hardware, and related subjects, including FORTH programming. Responsibilities include marketing seminar support and instructing in-house polyFORTH courses.

\section*{EDUCATIONAL STAFF ASSISTANT}

Experienced in dealing with public, sales and marketing, and some programming. Duties will include assisting education department manager with overflow administrative tasks, active participation in FORTH, Inc. user group.

\section*{JR. NSTRUCTOR}

Experienced in public speaking or educational instruction, programming on various processors --high-level languages and assembler. Microprocessor and FORTH programming background valuable.

CONTACT:
Pat Jones
FORTH, Inc.
2309 Pacific Coast Highway
Hermosa Beach, CA 90254
(213) 372-8493

\section*{CONSULTANT WANTED}

We are designing a heat pump controller system, which is based on the National Semiconductor "COPS" Microcontroller. It is a 4 bit calculator chip, with 2 K of ROM and 128 nibbles of RAM.

\section*{We need a consultant who can:}
1. Advise whether or not Forth can be put on the COPS
2. Estimate the program size, once compiled
3. Write software which would allow me to write and debug code on a TRS-80, Model I, and then cross compile it to the COPS.

For information call:

\footnotetext{
THE COLEMAN COMPANY, INC. Scott Farley Design Project Manager (316) 832-6545
}

\section*{NEW PRODUCTS}

\section*{FORTH by Timin Engineering, Release 3}

Release 3 of FORTH by Timin Engineering is : somplete software development system. It is -teractive (conversational) in nature. The = JRTH system incorporates a command processcr, compiler, editor and assembler, all memory esident. The principal benefits are a reduction
- software development time and a reduction
- memory size for large applications. The :-incipal application area has been machine and =:-ocess control. The language is suitable for all solications except scientific mathematics. - is product is based on the well-known FIG \(=J P T H\) but with numerous enhancements, - -luding:
- visual (screen) editor
- array handling (implemented in machine code)
- very fast disk I/O
- configurable for different memory size
- creates turn-key applications
- CP/M system calls and file handling

Qelease 3 of Timin FORTH will run on ZEC/8080/8085 hardware systems with CP/M or こכOS. Minimum memory size is 28 K . The : \(\cdot\).e for Release 3 of Timin FORTH is \(\$ 235\) (if rener than \(8^{\prime \prime}\) standard disk, add \$15). To order =eiease 3 of Timin FORTH, write Timin ミ-gineering Company, 9575 Genesee Avenue, S.ite E-2, San Diego, CA 92121, or call (714) -:5-9008.

\section*{HOOS FORTH}
- Vendor: Essex Computer Science
- Address: 1827 St. Anthony Ave., St. Paul, MN 55104
- Telephone: (612) 645-3345
- Contact: Rick Smith
- Product Name: Essex HDOS FIG-Forth
- Description:

Essex HDOS FIG-Forth is an inexpensive version of FIG-Forth for Heath

H89/Zenith Z89 users with the HDOS operating system. It is a version of 8080 FIG-Forth Version 1.1 customized for HDOS and the H/Z89. Disk I/O takes place via a standard HDOS disk file. In addition, the FIG-Forth source listings are provided and may be modified and reassembled on a single-disk HDOS sytem.
- Extras: None.
- Target machines: Heath H89 and Zenith Z89. Heath H8 users may also use the system if they modify the console I/O routines.
- Memory requirements: 32K of RAM
- Number of documentation pages: 140
- Documentation description:

Documentation consists of release notes, a copy of the FIG-Forth Installation Guide, and a copy of the official 8080 FIG-Forth version 1.1 source listing. The manuals provide the information necessary to install and modify the Forth system.
- Essex does not offer the manuals separately. They may be purchased separately through the Forth Interest Group.
- We will reduce the price to \(\$ 25.00\) for persons already owning copies of both FIG documents.
- Form of Product: 5" HDOS diskette, including source, object, and release note files.
- Shipments to date: about 4
- Price: \(\$ 45.00\), or \(\$ \$ 25.00\) for those who already own the FIG documentation.
- Includes: U. S. postage, local tax.
- Warranties and support: 30 day free replacement of defective media. We are interested in fixing bugs that crop up but do not guarantee that bugs will get fixed.
- Order turnaround time: 3-4 weeks.

Order from:

\author{
Essex Computer Science \\ Richard E. Smith \\ 1827 St. Anthony Avenue \\ St. Paul, MN 55104 \\ (612) 645-3345.
}

\section*{AN 1802 FIG FORTH}

Version 1--RCA CDOS
Load under RCA CDOS
Disc with source and object files for RCA CDP185008 CDP185007 CDP185005 with CDOS upgrade
A minimum of 8 K from address 0 is required
Version 2--RCA unit-track
Load under RCA unit-track
Dise with source and object files for
RCA CDP185008
CDP185007
CDP18S005 with UART card
A minimum of 8 K from address O is required
Version 3-object and FORTH screens
Load under RCA unit-track
then LOAD FORTH screens
Version 3 is suggested unless the user wants to manipulate the 1802 source code. This version will be continually updated with program material.

The discs are \(\$ 50\) each (Calif. res add 6 percent sales tax)
Order from: CMOSOFT, P. O. Box 44037, Sylmar, CA 91342

\section*{AIM-FORTH HACKER'S SYSTEM"}

I finally got my fig-FORTH 65 running on my AIM-65 at work and I would like to offer it to other hackers like myself. This FORTH system runs on AIM -65 with the DAIN DISK SYSTEM and uses an external terminal.

The software is on 2 disks. One contains the complete source and object. The other contains Editor, Screens, Error Messages and other bits of FORTH code of my creation done while I started using FORTH.

1 will supply my AIM-FORTH "Hacker's Syatem" to anyone for \(\$ 25.00\). THIS IS NOT FOR BEGINNERS! THIS IS NOT A COMMERCIAL PRODUCT! I am interested in contacting other FORTH hackers in my area and would like to possibly make some noise with them or start a phone line software interchange of techniques using MODEMS. I welcome any letters or input on this idea.

\section*{Eric Johansson}

55 A Richardson St.
Billerica, MA 01821
(617) 667-0137 (home)
(617) 899-2719 \(\times 224\) (work)

\section*{FORTH MAILING LIST FOR APPLE}

Allows users to maintain 1,000 entries per floppy. Functions include adding, deleting, and modifying entries. The mount option allows mounting any number of mailing list floppys. Labels can be generated in 1,2,3, or 4 across formats with user optional selection criteria.

This application package includes: 16 sector boot disk for the Apple; Source code for system and a bonus of one mailing list floppy with name addresses and phone numbers of over 100 FORTH users.

Price is \(\$ 45.00\) from:
Elmer W. Fittery
INTERNATIONAL COMPUTERS
110 McGregor Avenue
Mt. Arlington, NJ 07856
(201) 663-1580 (call after 6:00 pm)

FIG CONVENTION COMING - NOV. 28

\section*{FORTH CLASSES}

NEW CLASS
BY KIM HARRIS \＆HENRY LAXEN

\section*{FORTH，PRINCIPLES AND PRACTICES}

This class is intended to teach the student - －ow to write programs in FORTH．It is a＂how \(: \partial^{\prime \prime}\) class and not a＂why＂workshop．The class －ill meet on each Monday in October from 6：30 －2 9：30 at Berkeley Computer， 1569 Solano \(\therefore\) venue，Berkeley．The phone number there is ： \(26-5600\) ．The topics to be covered are：

The Language
Input Output Structure
String Handling
Vocabularies
Defining Wards
This is an ambitious schedule，and depending \(\Gamma\) the level of the students，more or less will be ：こvered．Experience with other computer lan－ コages would be helpful，though it is not equired．There will be homework exercises，三xd machines will be available for students＇ se．For more information，contact Henry ＿axen at（415）525－8582．

SEMINARS，WORKSHOPS，CLASSES
FROM FORTH，INC．
\begin{tabular}{lll} 
ocation & Seminar & Workahop \\
＿こs Angeles & October 15 & October 16 \\
汭 Diego & October 22 & October 23
\end{tabular}

Seminar

October 22

Workshop

October 23
and October 5－9 at FORTH，Inc．An advanced course will run October 12－16．Contact Kris Cramer for details．FORTH，Inc．， 2309 Pacific Coast Highway，Hermosa Beach，CA 90254， （213）372－8493．

\section*{MORE FORTH CLASSES}

Intensive 5 －day FORTH workshops are being offered at INNER ACCESS CORPORATION． These workshops provide an introduction to the FORTH programming language sufficient to design and debug programs to solve real problems．These workshops also serve to enhance one＇s understanding of the FORTH tools necessary for complex applications．
\begin{tabular}{lll} 
Workahop Dates & Time & Cost \\
Sept．21－25 & \(9-4: 30\) & \(\$ 295\)
\end{tabular}

Oct．19－23
Nov．16－20
To obtain more information on these work－ shops，call Inner Access（415）591－8295 in Belmont（home of Marine World）in the San Francisco Bay Area．

\section*{AND MORE CLASSES}

Free Beginner＇s Class for Apple users．In San Diego，two－session course on 9／26／81 and 10／30／81 at 1 p．m．at Computer Merchant， 5107 El Cajon Blvd．K．V．Amatneek，Instructor．

Introductory classes in polyFORTH ：－：ogramming will be offered September 14－18

How to form a FIG Chapter:
1. You decide on a time and place for the first meeting in your area. (Allow at least 8 weeks for steps 2 and 3.)
2. Send FIG a meeting announcement on one side of \(8-1 / 2 \times 11\) paper (one copy is enough). Also send list of ZIP numbers that you want mailed to (use first three digits if it works for you).
3. FTG will print, address and mail to members with the ZIP's you want from San Carlos, CA.
4. When you've had your first meeting with 5 or more attendees then FIG will provide you with names in your area. You have to tell us when you have 5 or more.
\begin{tabular}{ll} 
Northern California \\
\hline 4th Sat & \begin{tabular}{l} 
FIG Monthly Meeting, \(1: 00\) p.m., at \\
Southland Shopping Ctre, Hayward,
\end{tabular} \\
& CA. FORML Workshop at 10:00 am.
\end{tabular}

Southern California
Los Angeles
4th Sat FIG Meeting, 11:00 a.m., Allstate Savings, 8800 So. Sepulveda, L.A. Philip Wasson, (213) 649-1428.

\section*{Orange County}

3rd Sat FIG Meeting, 12:00 noon, Fullerton Savings, 18020 Brockhorst, Fountain Valley, CA. (714) 896-2016.

San Diego
Thur FIG Meeting, 12:00 noon. Guy Kelly, (714) 268-3100, \(\times 4784\) for site.

Northwest
Seattle Chuck Pliske or Dwight Vandenburg, (206) 542-8370.

New England
Boston
lst Wed FIG Meeting, 7:00 p.m., Mitre Corp., Cafeteria, Bedford, MA. Bob Demrow, (617) 389-6400, x198.

Boston
3rd Wed
MMSFORTH Users Group, 7:00 p.mos Cochituate, MA. Dick Miller, (617) 653-6136 for site.

Southwost
Phoenix Peter Bates at (602) 996-8398.
Tulsa
3rd Tues FIG Meating, 7:30 p.m., The Computer Store, 4343 So. Peoria, Tulsa, OK. Bob Giles, (918) 5999304 or Art Gorski, (918) 743-0113.

Texas Jeff Lewis, (713) 719-3320 or John Earls, (214) 661-2928 or Dwayne Gustaus, (817) 387-6976. John Hastings (512) 835-1918.

\section*{Mid Atlantic}

Potomac Joel Shprentz, (703) 437-9218.
Now Jersey George Lyons (201) 451-2905.
New York Tom Jung, (212) 746-4062.
Midweat
Detroit Dean Vieau, (313) 493-5105.
Foreign
Auatralla Lance Colline (03) 292600.
England FORTH Interest Group, c/o 38, Worsley Road, Frimley, Camberley, Surrey, GU16 5AU, England

Japan FORTH Interest Group, Baba-bldg. 8F, 3-23-8, Nishi-Shimbashi, Minatoku, Toyko, 105 Japan.

Canada
Quebec
Gilles Paillard, (418) 871-1960 or 643-2561.

\section*{West Germany}

Wolf Gervert, Roter Hahn 29, D-2 Hamburg 72, West Germany,(040) 644-3985.


Published by Forth Interest Group

Volume III No. 4

\section*{Publisher}

Editor
Editorial Review Board
Bill Ragsdale
Dave Boulton
Kim Harris
Jonn James
Dave Kilbridge
Henry Laxen
George Maverick
Bob Smith
John Sumgarner

FORTH DIMENSIONS solicits editorial material, commente and letters. No responsibility is assumed for accuracy of material submitted. ALL MATERIAL PUBLISHED BY THE FORTH INTEREST GROUP IS IN THE PUBLIC DOMAIN. Information in FORTH DIMENSIONS may be reproduced with credit given to the author and the Forth Interest Group.

Subseription to FORTH DIMENSIONS is free with membership in the Forth Interest Group at \(\$ 15.00\) per year ( \(\$ 27.00\) foreign air). For membership, change of address and/or to submit material, the address is:

Forth Interest Group
P.O. Box 1105

San Carlos, CA 94070

\section*{hSTORICAL PERSPECTIVE}

FORTH was created by Mr. Charles \(\mathrm{H}_{0}\) 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 cuatomers' unique requirements.

The Forth Interest Group is centered in Northem Celifomia. Our membership is over 2,400 worldwide. It was formed in 1978 by FORTH programmers to encourage use of the lenguage by the interchange of ideas through saminars and publications.

ORDER YOUR COPY!
Proceedings of the 1981 Rochester FORTH Standards Conference
\(\$ 25.00\) US, \(\$ 35.00\) Foreign. Send check or MO to FIG in US funds on US bank.
"Starting FORTH"
Hard Cover - \$20.00 US, \$25.00 Foreign
Soft Cover - \(\$ 16.00\) US, \(\$ 20.00\) Foreign

A apecial thanks this month goes to Mr. Larry Forsley and the Univeraity of Rochester. The majority of this issue comes from his offorts and those of his asociates. While acting as guest editor for this iasue of FORTH DIMENSIONS, Mr. Forsley was also compiling and editing the proceedings from this year's FORTH conference at the University of Rochester. Even with this "double duty," Mr. Forsley has done an excellent job.

The quality of material we have received from the University of Rocheater is excellent and greatly encourages me in my plans to "de-Californize" FORTH DIMENSIONS through the use of regional guest editors. While Mr. Forsley and the University of Rochester may be tough act to follow, I will welcome contacts from anyone else (person and/or organization) who would like to try guest editing an issue. For your peace of mind, let me assure you that production (typesetting, proofing, printing, etc.) will be handled for you. If you think you have what it takes, give me a call or drop me a line.

You may find that some of this issue's sections have been reduced is size and/or eliminated. This is a temporary concession because of the volume of material we have to publish in this issue. Postal costs prohibit expanding the size of FORTH OIMENSIONS to publish all we receive, so when we have a quantity of quality material we publish those items that would seem to have the greatest reader interest.

I hope to meet many of you at the FIG National Convention in Santa Clara, California on November 28th. Meanwhite, GO-FORTH and get additional members.

\section*{C. J. Street}

Editor

\section*{PUBLISHER'S COLUNN}

We are heading into some busy times for FIG. By the time you gat thia copy of FORTH DIMENSIONS we'll have completed the Mini-Micro Show in Southern California and be deep into the detalls of the FORML Conference and FIG National Convention. Remember that the Convention is Saturday, November 28th at the Marriott Hotel in Senta Clara, California. Expect to see many of you there.

We've sent out packets to FORTH vendors about exhibiting at the FIG National Convention. If you are interested in exhibiting and haven't received a packet, call the FIG line and request one: (415) 962-8653. Only \(\$ 50\) for a table!

This insue is the much awaited University of Rochester effort. Its packed with useful material. You ought to order the Proceedings of the 1981 Rochester FORTH Standards Conference. It has 378 pages of excellent papers
"Starting FORTH" by Leo Brodie is available from FIG --------m--- and replaces "Uaing FORTH" as the book to have about the FORTH language.

Now, little lecture. We have conducted en unscientific survey and found that in many locations there are people who are using FORTH and aren't members of the FORTH Interest Group. You as a member should work on them to join. All you have to do
 heve your asociates fill in their name and adaress. If we each get one more person to join we'll have over 5,000 members. Let's do it.
Roy C. Martene

\title{
FORTH AND THE UNVERSTTY
}

\author{
Lawtence P. Forsley \\ Laboratory for Laser Energatics University of Rochester
}

Welcome to the wonderful world of FTH, or, University of Rochester - JRTH. URTH wes developed several - ears ago and has been used for many wolications, some of which are sosumented here. Beginning with the \(=-8\) FORTH Internatina! Standards Emnierence, held on Catalina, we have :-.owed the FORTH standardization -"ort. As a result, the majority of our sistems are close to being FORTH-79 E:sndard, although not FIG model. Very - - w papers in this issue will refer to - © TH.

The 1981 Rochester FORTH Standards Eonference was held at the University. - major reason for this, aside from the e.gntful weather at that time of year, is \(\because\) FORTH activity at the University. --.s work shows up in several divisions and zeartments including the University =:mouting Center; Optics; Physics and -s:-onomy; Chemical Engineering; \(\therefore=\) Ehanical Engineering; Department of = ¿.ology, Division of Diagnostic Ultra-si=- d; Department of Cytopathology; Eertical Engineering and the Laboratory \(\because\)-aser Energetics. Indeed, we are rested to the original work by Dick \(\vdots-7\), who in 1976 was an assistant profes-z- If Physics and Astronomy, for deriving - first URTH system; and to Ken -a-Iwick, who in 1977 was with the -r.versity Computing Center, for bringing \(=\) :ne IBM 360/65 TSO version based on E Ex's work. At this time, Ken, Dick and I -E:e the only FORTH users at the - ersity. I believe the name URTH was : ed by Ken, although Dick was partial PARTH, for Mike Williams' -."tasking Intel 8080 FORTH system. --' rrtunately, Ken and Dick are no longer -:- the University; and Mike's commit-- -ts prevented his authoring a paper. Toever, their work is reflected in the -a:erial presented here.

This issue starts with three overview zicars. The first paper is mine and covers -e development of FORTH at the Labora\(\because\). for Laser Energetics, which remains -e largest university FORTH user. The *: and paper, by Peter Helmers, reflects \(\tau\) :he uses of FORTH in medical research Fc elinical applications. The third, by ren Lefor, covers one of the more visible r.ersity FORTH systems: The IBM 3032 \(-\approx=0 m m u n i c a t i o n s\) front-end.

The next three papers demonstrate a \(\mathrm{i}^{-}\).ety of ways by which FORTH can be se: to interact with hardware. The first zHe?, by Rosemary Leary and Carole * aler, deals with three methods of using -acsed memory. A second paper, by Bob

Keck and me, demonstrates a high level interrupt handler used in plasma physics experiments. The third paper in this section is by Joe Sawicki, and suggests powerful structures for easily and efficiently interfacing hardware.

The last section Illustrates the difficulty with defining the difference between systems and applications. The first paper is by Michael McCourt and Richad Marisa, and describes a transportable String 5tack. The second paper is by Alfred Clark and covers a FORTH-based complex arithematic calculator. The last paper is by Greg Cholmondeley and documents a microprocessing tool similar to one supplied by Signetics.

These papers have many things in common. One example is the difficulty in discriminating between users and implementors. Bob Keck, a user, worked with me to develop a tool for high level interrupt handling. Likewise, Al Clark, also a user, has augmented a flosting point package with words appropriate to the complex plane. The String Stack is clearly a system tooi. Complex arithmetic is leas so, and a microprogramming system is clearly an application. Or is it? In the context of its user, the microprogramming words are a system. We seem to be forever chasing our tail when determining a FORTH context. But 1 think that this is the power of FORTH.

Another facet is the use of defining words used throughout the papers. An extension of defining words, Paul Bartholdi's TO concept, is used in both Joe Sawicki's and Greg Cholmondeley's code. Mike McCourt's "IN" concept \({ }^{2}\) is used by Peter Helmer's to implement the TO concept. However, a student, Carole Winkler, thought that TO complicated things unnecessarily, so she doesn't use it.

This last comment illustrates one of the virtues of universities: freedom of dissent. Unfortunately, I have found that most groups, and many people, using FORTH are intolerant of different views. During my involvement with FORTH I have watched many groups rise to ascendency, tout the true way, and then be replaced by another group. This has been especially true of the FORTH Standards effort where Kitt Peak, FORTH, Inc., the Europesn FORTH User's Groups and FIG have all played this role. But another view is possible, which is more in keeping with FORTHE nature.

Many of us see FORTH as being' a system of controlled, or directed, anarchy. Since every man, or woman, can be for himself it is highly idiosyncratic and anarchistic in form. Anyone who has tried a team approach to FORTH programming is familiar with the tendency towards a Tower of Babel. On the otherhand, people comfortable with thie
unstructured environment find both their productivity and creativity increased. But, some direction must be applied to share code smong users. I suggest that this direction should be one of form, and not of content.

It is appropriate to define documentation atandards which imply a form. But is is inappropriate to state that something can be dane only one (with the implied right) way. However, people who learn something by doing it the wrong way understand much better than people who are told the right way.

I think an example of this can be foung in a conversation I had with Kim Harris.' Kim took exception to an enrlier paper by Peter Heimers on Userstacks. \({ }^{4}\) I was told that the approsch was wrong. Period. But on further discussion, 1 found that 1 agreed with Kim. The fault was that Peter had found only a partial solution to data typing, and in a multitasking system his technique might be very cumbersome. That's fine. Peter Helmers does not use multitesking systems, as his systems are all single user, interrupt/event criven. thus, it is worth remembering that each of us has different, and valid, viewpoints.'

As a major promoter of FORTH at the University of Rochsater, I have tried to define en environment conducive to this type of interplay. This has resulted in a learning environment with many student opportunities; and with Leo Brodie's book, Starting Forth, and Don Colburn's study guide, Going Forth, we can begin teaching with FORTH. Not teaching FORTH, but teaching with it. Four of the authors in this issue are students and three other authors teach courses or seminars. If FORTH is ever to catch on like Pascal, or FORTRAN, then it must begin wtih university teaching as those two languages did. In five years my present students will be in industry, as my first student contacts slready are. A univeristy environment coupled with its students' enthusiasm and their eventual employment will further FORTH more than any seminar series or interest group. But it will take time.
1. FORTH DIMENSIONS Vol. I No. 4 and Vol. I No. 5.

\section*{2. FORTH DIMENSIONS Vol. II No. 4}
3. Personal conversation on May 10, 1981 prior to the Rochester Conference.
4. FORTH DIMENSIONS Vol. II, No. 2
5. Since that paper, Peter has published another one, entitled "Alternative Parameter Stacks," which can be found in the Proceedings of the 1981 Rochester FORTH Standards Conference.

\section*{FORTH N LASER FUSION}

Lawrence P. Forsley
Laboratory for Laser Energetics
University of Rochester

\section*{Abstract}

Inertial confinement fusion research using lasers has resulted in the laboratory creation of extraordinary conditions of temperature and pressure, duplicating those found in the cores of white dwarf stars. The machines which create these conditions and the diagnostics that monitor them have become increasingly automated. The demands of this research have forced us to adopt new tectmiques, like FORTH, for enhancing interactions between engineers, physicists and their experiments.

\section*{introduction}

Lasers have been used to simulate plasma conditions of high density (approaching solid) and temperature (over 60 million degrees) for several years. The goal of these experiments has been either for weapons effect simulation, practiced at the national laboratories, or for the possible commercial generation of power. This latter program has been exclusively pursued by the Laboratory for Laser Energetics (LLE) for almost a decade. As can be expected, these experiments have resulted in the development of new diagnostics, and these diagnostics, in tum, have resulted in new fields of physics. Besides the Laser Fusion Feasibility Project, there are research programs in: sub-picosecond lasers, nanosecond X-Ray sources, X-Ray lasers, laboratory astrophysics, and materials damage testing.

These research programs, and the main supporting lasers, are highly automated. About one half of the computer systerns on the 24 beam 13 terrawatt infrared Ornega laser and all of the computers on the single beam Glass Development Laser (GOL) are implemented in FORTH. This paper will explore the development of FORTH-like languages at LLE.

The laboratory is also part of the College of Engineering of the University of Rochester. Thus, there is an important interplay between the staffs, and atudents, of LLE and the University. Most of our FORTH systems have been partially, or totally, implemented by students from chemistry, electrical engineering, physica and computer science. Four of the other papers in this journal issue have a student author who is also a member of LLE.

\section*{Standardization}

LLE was one of the first Laser Fusion laboratories to automate its laser systems. Whenever possible, we relied
upon standard computers, interfaces and software. Originally, in 1971, we chose the Hewlett Packard 2100 series computer, and the RTE (Real Time Executive) Operating System with Fortran, Assembler and Algol. We used the HP backplane for our instrument interface. This system ran for over five years and 15,000 shots, but building a completely automated laser with 24 instead of 4 beams required a different approach.

The Hewlett Packard computer backplane was limited in the number and variety of devices which could be procured and attached to it. We overcame this difficulty by adopting CAMAC ( 5 ). CAMAC provided us with a large capacity, computer-independent backplane. It was also a widely used standard in the nuclear physics community with instrumentation and interfaces appropriate to our needs available from several sources.

The problems of computer and software standardization were more difficult. Some of our applications were realtime, and appeared to require a fast interrupt response. In other cases, we were interested in direct image digitization and needed a large address space. Other requirements suggested the need for a powerful multiprogramming operating system. Unfortunately, no one computer type and operating system supported all of our applications; and yet, with limited manpower, it was difficult to support a variety of hardware and software.

Computer languages, including FORTRAN, are different from one vendor to another, and especially when operating system calls were taken into account. The problam of software consisteney and support was not limited to dissimilar computers. Ehrman ( \(4: 16,17\) ) has shown that as many as 12 different languages may be encountered by a programmer when editors, linkers, and loaders are included in addition to the programming language. Therefore, unifying software approach was needed among various operating system functions and languages on the same and different computers. We did not know of the unix System from Bell Laboratories (11:1905-1929) and the ' \(C\) ' programming language of Richie and Stevens (12:19912019) in 1976. However, I had talked with people at Kitt Peak in 1976 and travelled there in the spring of 1977 to see FORTH being used.

\section*{FORTH}

FORTH was originally developed as a small, real time operating system for telescope control and image processing by Moore (8:497-511), (9) and Rather (10:223240) at the Kitt Peak and NRAO facilities which are funded by the National Science Foundation. Ifound three groups at these facilities using FORTH scientists, computer engineers and techniciana. In ame
cases, the scientists were very knowledgeable about FORTH, whereas in other cases, they only knew a few words. I was especially impressed by Dr. Mark Alcott, who was, at the time, with Cal Tech and was observing on NRAO's 36 foot radio telescope. He was pleased with his ability to change the graphics routines and other "systems" software while cantinuing to collect data. Similarly, 1 found many technicisns programming and writing test programs. This appeared to make good use of their time, especially when they would be familiar with a device, like a Varian computer disk controller, and did not have to explain its function to a programmer. It also appeared that many of the computer group's staff enjoyed FORTH, although there were problems witr standardization and change. I found out several years later, talking with Jeff Moler, who was then in operations at Kitt Peak and is now with the Livermore Tandem Mirror Experiment, how difficult it was to meintain programs in this environmant.

FORTH seemed to have many desirable characteristics, and it provided the same programming environment on many machines. It allowed both very low level access to hardware and high level structures to shield users from that hardware. There was an assembler, a compiler, and an interpreter. What we did not know then was the care required in dacumenting it, and the tendency to create personalized applications and words. But, we needed a version of FORTH at the University.

Dick Berg, an assistant professor in physics and astronomy at the time, \({ }^{2}\) decompiled a Kitt Peak Varian nucleus circa 1974. He recoded it for the National Semiconductor PACE microprocessor. Ken Hardwick, then with the Univerity Computing Center,' used this as a modei for the IBM 360/65 under T50 and Mike Williems developed multitasking version on the INTEL 8080. This was the birth of URTH.

We also procurred a version for the Zilog Development System from FORTH, Inc. at about the same time to demonatrate an automated X-Ray spectrometer. Although 1 had a system for the Hewlett Packard 2100 from Kitt Peak and a "diskless" version from Don Berrian at Princeton, I decided that we should develop our own version based upon the URTH model. Ken Hardwick and I did this in late 1977. Since then, other members of the Univeraity community and the Laboratory for Laser Energetics have worked on various versions of FORTH for Date General, Modcomp, PDP 212 and IBM 3032 computers. Through the efforts of Mike McCourt, originally with the Department of Cytopathology and then with LLE, we developed FORTH-79 system. All of these were multitasking systems (2:314-

The first FORTH applications at LLE -ere hardware testbeds. There are two :stinct phases in dealing with hardware. - he first occurs during its initial checkout and reoceurs when it fails, or you suspect : of failing. At this stage, one is conserned with device and interface implementation, and it is important to be able \(i s\) interactively set and test data and adsress lines.

A testbed must be capable of exerising hardware at a rate of ebout 1 kiloertz. Devices which operate in a faster :.me domain will usually be buffered, as m example, with transient digitizers. Aost other devices, such as relays, seerate in 10 Hz or slower time somain. At a 1 kHz rate, sufficient samz.es can be taken from \(A / D ' s\) and \(D / A^{\prime}\) s to suckly check their accuracy and range, and thereby checkout many parts of a sysem quickly.

Several language features are required ' r tests like these. A means must be proised to individually and collectively set aedress and data lines. There must also be 3 way of repetitively issuing data/ address =atterns. Often, a hardware problem is -termittent, and a test and branch capa:.ity is necessary to allow loopiung until a 'allure occurs.

Thus, the specification for a testbed anguage grows quite large, with a major - ie occupied by the command processor, \(x\) text interpreter. Regardless of नether the testbed language is imple--ented in Fortran, Basic, Pascal or most :iner programming languages, a substan:.a! effort will be spent on the text inter:reter. One of the virtues of FORTH is :-at it comes with a generalized text terpreter, suitable for testbeds and : : eer applications.

Our FORTH testbed applications in:.Jded: power conditioning testbed for =-ecking out laser amplifiers; alignment :estbed for debugging and calibration of *-tomated components; " and, general こ 2MAC module testing. Other testbeds -ave been used to develop image prosessing hardware and software, and oneamensional reticon arrays.

The laser amplifier testbed was seveloped along the following schedule:
1. October 1977-Ken Hardwick and I began writing a FORTH system for the HP 2114.
2. Jenuary 1978- The FORTH system was completed and CAMAC software started.
3. March 1978- A laser amplifier testbed was demonstrated.
4. April 1978- Single laser amplifier tentbed was operational at laser hardware subcontractor's site, with a duplicate at LLE.

By April, it was clear that the Omega Power Conditioning computer would not be available until August, 1978. Since the Department of Energy four-besm milestone was originally scheduled for early September, 1978, this left insufficient time for laser preparation.
5. April 1978- An LLE engineer, John Boles, and a consultant with the software subcontractor developing the power conditioning software, began coverting the single amplifier testbed to run 4 laser beams synchronized with the laser oscillator.
6. June 1978- A six beam laser aystem was operational.
7. August 1978-Preliminary delivery of full 24 beam system which was Fortran-based.
8. October 1978- Department of Energy Milestone passed.

There were substantial differences between the 24 beam Fortran based system and the 6 beam FORTH version. These included the lack of an error detecting command processor, a graphic display and error archiving on disk. However, whereas the FORTH version used 16 K words of memory and floppy disk, the Fortran based system required 196 K words of memory and a 15 megabyte hard disk.

This application also made us aware of FORTH'S compactness and the speed with which applications could be developed. It is my feeling that this, and several other applications, were brought up in one half the time it would have taken in Fortran, including FORTH training time. Once good documentation is available, FORTH will prove even better.

Also, 1 have found FORTH systems to be more maintainable than comparable Fortran systems, because FORTH uses 10 times fewer source lines. Some care is needed when writing FORTH. Another advantage can be gained by the ease of using data base technology when building process control systems in FORTH.

\section*{Spatial and Temporal Relationshipa}

The first phase of dealing with hardware is over when the hardware works. The relationships among devices then become important. One can hierarchically
organize related devices into aubsystems. This hierarchy consists of both spatial and temporal relationships among components (1), (3). The manipulation of these relationships requires the development of a data-base-like language. My initial work with Fortran and RTE, and discuasions with Ray Helmke and Eric Knobil at the Wilson Synchrotron, \({ }^{4}\) led me to develop such a language for process control called Maps, because it "maps" relationahips 6:109,110.

A Map contained two types of atructures, or Tags. A tag was either a collection of data, or a set of pointers to other Tags. The Map contained an inverted list of pointers to each tag, so that all tags were unique and accessible. Two specialized programs, SETUP and BUILD, were developed to manipulate and create the initial Maps from text files. About a dozen subroutines were developed to allow tags to be accessed. Data could then either be placed into one or more Tags, or retrieved from them. In the interest of speed, this system was recoded in assembly language and later microcoded on a Hewlett Packard \(21 M X-E\) computer. This computer currently suns the Omega 24 beam power conditioning, and was mentioned in the Testbed Section of this paper.

Alternatively, by using the text interpreter and FORTH's capability to define arbitrary data structures, several data-base-like systems have been developed. In its simplest form, everything in FORTH is an executable data structure. Thus, FORTH allows one to define spatial and temporal relationships in a simpler, and more concise fashion than Maps. In addition, it is internally consistent, whereas Maps had Fortran, assembler, microcode and operating system interface facets.

\section*{Production Syetarme}

Once FORTH had proven viable for small systems, we decided to implement production systems in it. These systems included automated diagnostics as well as the laser control systems. The prototype Omega 24 beam calorimetry system was an example of an early production system. It used simple, vector like structures to contain the addresses, relationships and values associated with various calorimeters, analog to digital convertors and calibrators. It was capable of displaying beam energies and calculating exponential fits to the data.

The Omega 24 beam Alignment System is more complex. It has run on an LSI 11/2 with 5 CAMAC crates and 3 color displays, controlling over 1000 devices. Initially, the operators used the FORTH text interpreter for all commands and queries. One advantage was their ability to write new "macros" to setup complicated slignment procedures more quickly. However, there was a risk asso-
ciated with letting operations' personnel directly program the system. Therefore, the new Alignment System has a more complete command processor implemented in FORTH, but which does more error detection than the simple text interpreter. This system also uses the defining words capability and has a large disk resident data base for describing components. With the advent of the command processor, the system was switched over to an LSI 11/23 with mapped memory. This addition allowed approximately 20 tasks to handle various functions, communicating via a queue-based message protocol.

The laser beam quality is also important to us. We use streak cameras interfaced to Princeton Applied Research Optical Multichannel Analyzers for this purpose. The PAR OMA includes a FORTH-based LSI 11 for acquisition and reduction. As with the early Alignment and Calorimetry systems, it is programmed directly in FORTH. \({ }^{6}\) Unlike those systems though, this was originally not a tumkey system provided by software engineers, but rather was incrementally developed by physicists and students.

We also use FORTH exclusively on the Glass Development Laser (CDL) with similar computer systems. A FORTH based HP 2100 is used for power conditioning and interlocks for the main bay and three surrounding laboratories. A DEC LSI 11/2 collects laser and target calorimetry data, reduces it, and also maintains a data base on disk. A second LSI II is used in a PAR OMA for processing streak camera data. This is especially significant since GOL is engaged in converting the infrared light to ultraviolet, and the first harmonic \(\mathbb{R}\), a second harmonic green and the third harmonic, UV are observed with the same streak camera. This required very flexible system to allow reduction in a quasitwo dimensional mode. Another Hewlett Packard 2100 has two video digitizers and a color graphics unit. It is used for determining absolute beam intensity and modulation for materials damage testing. This system is being converted to \(a\) DEC LSI 11/23 with an RLO1 disk attached. A third LSI 11 has been used by a graduate student, to observe target plasma produced X-rays. Finally, an LSI \(11 / 23\) is used with the nanosecond X-Ray facility for the real time acquisition and reduction of 2D X-ray diffraction patterns. Recently, this system has had an array processor interfaced to it to allow real-time fast fourier transforms of sample diffraction rings. All of these systems are FORTH based, with the automated imaging diagnostics serving as prototypes for Omega diagnostics.

\section*{Conclusion}

Although FORTH was relatively unknown, it has made a positive impact on the development of systems and instru-
mentation at LLE. It has allowed the computer sytems group to sdopt the philosophy of providing tools to scientists and engineers, equipping them to do job themselves. Sometimes, it was questioned whether this was the best use of their time: and, for some people, it wasn't. But, for the majority of people in GOL, and a fair number on the Omega syaterns and other laboratories at LLE, FORTH has been a success.

\section*{Acknowledgements}

I would like to thank an almost endiess list of people for their help over the past five years. Most importent among them though, are Ken Hardwick, Dick Berg, Chip Nimick and Mike McCourt. Also, without the help of many students during this period, many of these syterns would never have been built.

This work was partially supported by the following sponsors: Exxon Research and Engineering Company, General Electric Company, New York State Energy Research and Development Authority, Northeast Utilities, The Standard Oil Company (Ohio), the University of Rochester, Empire State Electric Energy Research Corporation, and the U. S. Department of Energy inertial fusion program under contract number DE-ACO8800P40124.

Lawrence P. Forsley is group leader of the Computer Systems Group at the Leborstory for Laser Energetics, University of Rochester, Rochester, N.Y.

\section*{Footnotes}

1 The four-beam system, Delta, had computer control and monitoring in 1972. (6:101).

2 He is now with the Defense Mapping Agency in Waahington, D.C.

3 Kan is now with Network Systems Inc., in Minneapolis, MN.

4 Comell Univerity in the summer of 1977. This facility is now known as the Cornell Electron Storage Ring.
5 The mapped memory tectmiques are discussed by Leary and Winkler in the "Mapped Memory Techniques in FORTH" paper in this issue.

6 PAR purchased this system from FORTH, inc.
7 This is mentioned in Bob Keck's and my paper, "A High Level Interrupt Handier in FORTH", which can be found in this Issue.

\section*{PROCEEDINGS OF THE 1981 ROCHESTER FORTH STANDARDS CONFERENCE}

Many have been waiting for this conference proceedings to come out, from what was a very interesting, and different conference. It was the first conference to address the FORTH Standard since the Cataline meeting of October 1979. Although it was suggested that the Rochester conference was only a regional meeting, attendees came from six countries and thirteen states. Also notable, we successfully divided papers into serial oral sessions one morning and had parallel poster sessions that afternoon. This way, almost averyone of the seventy participants presented something, and no one missed anything (we think).

In addition, we added travel spansorship this year. The Standard Oil Company (Ohio), Friends Amis, Inc., Miller Microcomputer Services, and Software Ventures contributed over \(\$ 5,000\). This travel fund covered partial travel expenses for attendees from as far away as Hawaii, Chile, Germany and the Netherlands, and as close as California and Kentucky.

The original call for papers was in three major areas: the Standard, floating point and files management. These areas are well represented in the proceedings. In addition, there are sections on Philosophy, Vocabulary, Multi-tasking and Data Acquisition, Data Structures and the Future of FORTH. The organization we sdopted combined poster sessions, oral sessions and sorne material not presented at the conference. There is an entire section devoted to working groups on areas like Standards clarification, FORTH techniquet, Floating Point and Files Management. There are 378 pages covering the state of FORTH. The Proceedings are avallable for \(\$ 25\). See the FIG Order Form.

For those who are interested, there will be another Rochester FORTH Conference the third week of May, in 1982. The tentative subject ares will be Process Control and Data Acquisition. We expect that there will be subareas dealing with microprogramming, FORTH machines, personal computing, and the Standard. For information, please contact the conference chairman:

Lawrence P. Foraley
Leboratory for Laser Energetics
250 East River Road
Rochester, NY 14623

\title{
IMPLEMENTING FORTH BASED MICROCOMPUTERS AT THE \\ UNVERSTTY OF ROCHESTER MEDICAL CENTER
}

Peter H. Helmers

\section*{sntroduction}
"The micros are coming"' Everyone -as heard this so that it is not unexpected :ᄀat physicians and researchers at the University of Rochester Medica! Center ask the question: "How can they be put to se?" Over the past four years l've been a:tempting to answer this question by assembling a series of microcomputers for zoth research and clinical applications. -hese systems are all similar in their use :? an S-100 bus hardware architecture and ; FORTH software environment. Yet they i:ffer significantly when it comes to secific hardware interfaces, application software, and types of system users.

In this article, 1 am going to focus on :oth these similarities, and these differences in microcomputer systems. I am ;aing to start out by discussing their :ommon hardware foundation, and then explore peripheral devices unique to each :-stem's design. Because the ultimate -sers of a system have a significant mpact on application software, I am going :: try to characterize the types of users 1 -ave dealt with, and their specific soft--are capabilities and needs. From here 1 - Il discuss some common software packajes that were written to transcend both -ariable hardware, and variable user, - \(\because\) quirements. By discussing all of this in e:ns of how FORTH has aided system sevelopment, I hope to fully support my : entention that FORTH is an ideal envi--nment to meld many different types of -sers to just as diverse hardware configu--ations.

\section*{Seneral Herdware Organization}

So let's start out by considering the :=mmon architectural arrangement of -ese microcomputers. They are all Z -80 : ased machines with typical memory sizes :' from 32 K to 48 K bytes of static read/ --:te memory and IK to 2 K of EPROM -enory used to contain machine specific mplementations of commonly needed I/O Zutınes such as console and disk drivers. Ezeh microcomputer uses one or two eight -sh single density floppy disk drives. The =-mary system console is comprised of a \(\therefore\) line by 64 character memory mapped Jeo display along with detached ASCII - \(\because\) board. Each machine also has an RS\(\therefore: 2\) serial port for printer hookup.

These computers are all organized - ound the 5-100 (IEEE-696) bus with from on to fifteen card slots available. With :-e basic setup described above using from Sur to six of these slots, the customiza\(m\) to specific system configurations is
accomplished by mixture of standard commercial and/or wire-wrapped paripheral interface cards. Let's consider some of these syatems in greater detail, looking at special hardware and how this is reflected in the systems' software.

\section*{Ultrasound Diffrection Apparetus (UDA)}

The UDA microcomputer is part of an experimental system to explore the scattering (diffraction) of medical ultrasound signals through tissue samples. The scattering is a function of both frequency of the ultrasound signa! ( 2 to 8 Mhz ) and the angular position of a receive tranaducer relative to the ultrasound transmitter. The UDA system thus must control three primary functions: analog carrier signal generation, tissue sample positioning, and received signal analog processing. At present, only sample positioning (using stepper motors) is not directly handled by the UDA microcomputer.

Carrier signal generation is controlled by means of Hewlett-Packard 8165A programmable aignal generator interfaced to the microcomputer by means of an IEEE-488 (GP-IB or HP-IB) instrumentation bus. An opto-isolated parallel TTL output port is used to control a programmable attenuator on the output of the 8165A. With a range of 0 to 130 db , the attenuator can be used to automatically adjust gains for maximum signal dynamic range.

The most critical aspect of the UDA hardware is the generation of gating signals used by the enalog processing circuitry. This is accomplished by using high speed analog mixers driven by digital timing circuitry with a resolution of 100 nsec., and an accuracy of \(0.01 \%\).

\section*{Study of Vein Mochanics}

The basis of this system is an experiment to measure axial force, diameter and transmural pressure in a blood vein (in vitro) while controlling axial strain and pressure. The system consists of a vertical chamber for the vein specimen, a prefusion and pressure clamping apparatus, force and preasure transducers, and a microprocessor for data acquisition.

The microprocessor contains a sixteen channel, twelve bit multiplexed analog to digital (A/D) converter to digitize the force and pressure signals under high level program control.

In conjunction with this A/D is a commercial video (TV) digitizer capable of programmed resolution ip to 240 lines of 256 picture elements. The input to this digitizer is from a TV camera aimed at the blood vessel under study. A special code definition was written to analyze a programmable area of the TV image for an indication of vessel diameter. This works
by first threshholding, then detecting vessel edges via a software algorithm. By using FOR TH/Z-80 assembly language, the dismeter determination executes in leas than one second.

This data acquisition system also contains a dual mode graphics display capable of \(128 \times 128 \times 4\) grey scale images or \(256 \times\) 240 dot graphics. Digitized video images use the former mode while acquired pressure and force data use the dot graphics. In addition, the TV signal dynamic range can be studied by adot graphic plot of TV signal amplitude versus time.

Also included in this syatem, to aid in data reduction, is an Advanced Miero Devices AM9511 high speed floating point processor IC. This circuit's speed, combined with the memory mapped graphics display, allows real-time analysis and display of acquired data, thus giving continuous feedback on the progress of the experiment.

Overall, this system replaced a manual strip chart and photographic recording setup that required several days for deta collection and analysis. Now data can be automatically acquired and processed within a couple of hours.

\section*{Pulmonary Microcomputer}

The pulmonary clinic uses a microcomputer identical to that just described except without the TV video data acquisition interface. Used in a clinical setting, this pulmonary microcomputer is integrated with a mass spectrometer and a breathing chamber to allow analysis of pulmonary tissue volume and capiliary blood flow. The basic procedure requires keeping track of the patient's breathing (by monitoring volume within the flexible breathing chamber) while analyzing the decreasing concentration of two soluble gases: dimethyl ether (DME) and acetylene \(\left(\mathrm{C}_{2} \mathrm{H}_{2}\right)\), referenced to the concentration of an insoluble gas: helium (He).

The hardware floating point unit facilitates rapid ( 30 seconds) analysis of the acquired data, including several curve fitting operations, and analysis of signals for relative maxima/minima. The graphics interface allows immediate viewing of the acquired data to ascertain proper signal levels, and to compare raw data to the curve fit data.

\section*{X-Ray Scanning Syztem}

This experimental scanner uses a slotted wheel and two horizontal slots (mounted at \(90^{\circ}\) to the radial orientation of the wheel) to achieve a mechanically raster scanned \(X\)-ray source. The wheel and horizontal slots are controlled by means of three separate stepper motors pulsed under control of the microcomputer. X-ray exposure is also
controlled by the computer as a function of measured patient \(X\)-ray attenuation.

The microcomputer contains a counter/timer chip which is used to control the stepper motors, a seven channel multiplexed eight bit \(A / D\) converter (used to measure patient \(X\)-ray sttenuation and \(X\)-ray power), and an eight bit D/A converter to control the exposure time of each \(X\)-ray pulse. Several digitai I/O lines are used to start the \(X\)-ray rotor, turn on the \(X\)-ray generator, and control stepper motor direction. Other lines are used to sense mechanical limit switches.

The software used in this machine is primarily concerned with controlling exposure time for each X-ray pulse in synchrony with the motor movement. The system ramps the motors up to speed from an initial stopped condition. In addition, it gradually increases speed to compensate for linear speed as the horizontal slots are moved radially towards the center of the wheels. The software also controls exposure time by sampling the attenuation of X-rays through the patient once each motor step, and using table look-up techniques to set the next pulse's exposure time. In addition, total x-ray power is sampled and accumulated to keep track of total patient dosage and \(X\)-ray tube usage.

\section*{How Uners' Needs Impect These Syatoms}

In my development of these systems, I have encountered three types of users: system developers, researchers, and physicians (and their clinical technicians). This grouping of users also roughly corresponds to levels of FORTH software utilization. The system developer-myself and presumably yourself--is expected to know all the in's and out's of system operation. If something is missing, it's generally easy to add it; this is a primary reason why many of us like FORTH. However we don't actually apply a system, we only set up the software foundation for the system. As users, we don't count.

A true end user, whether researcher or physician, cannot be sold on FORTH because missing capabilities can be easily filled in; they don't have the knowledge to do so. Nor do they really want to learn to do so. They have to be sold on other virtues of FORTH.

In my experience, researchers have been very receptive to FORTH. In general they have sophisticated technical backgrounds but little practical computer knowledge. This is a prime benefit: they may have used FORTRAN on a large machine for number crunching, but otherwise they have fow preconceived notions about computer arganization. They are less impressed with structured programming techniques or file syatems than they are by the fact that they can physically,
and interactively, control peripheral devices. A research scientiot may not understand how word like RAMP or SAMPLE works, but can readily learn what they do.

For example, the FORTH software written for the UDA sytem allows explicit user control of the hardware for setup purposes as well as automatic control during experimental data acquisition runs. Setup can be done through words such as:

\section*{OK 25 DB}
( RPN's a natural herel)
OK " PRQ 2500 KH2" TALK
(via the GP-IB)

\section*{OK 2.5 USEC CARRIER-OFF}

A data acquisition experiment can be set up using words such as:

OR 1002000 SWEPT-FREquexcy
( define control of ap8165A) OR FIXED-ATTEANUATION
( define control of atten) OR DON'T-SHOW-ATTENUATIONS \(\overline{O R} 150032\) NOVA-CONRROL
( let the minicomputer take over control of the micro.)

In addition, the researcher can build upon basic words to create custom application programs as needed. Thus the \(X\) ray scanner system can be easily programmed by:
\begin{tabular}{|c|c|}
\hline \multicolumn{2}{|l|}{\multirow[t]{2}{*}{OR MOTOR WHEEL-MOTOR ( define a 'MOTOR' data type)}} \\
\hline & \\
\hline \multicolumn{2}{|l|}{OR : ROTATE-EM} \\
\hline \multicolumn{2}{|r|}{D} \\
\hline \multicolumn{2}{|l|}{OK WhEtch-MOTOR RNMP} \\
\hline \multicolumn{2}{|l|}{( ramp stepping motors)} \\
\hline \multicolumn{2}{|l|}{OX LIMIT-SMITCHES?} \\
\hline \multicolumn{2}{|l|}{( exit loop if motor limited) OR SYNCERONI ZE} \\
\hline \multicolumn{2}{|l|}{( synchronize to motor pulse)} \\
\hline OR & toop \\
\hline OR & \\
\hline
\end{tabular}

A physician or clinical technician is much more of an end-user than the researcher. As such, they are less concerned with words that allow them flexibility in control of peripheral hardware; instead they want words that control hardware in epecific ways towards some specified clinical objectives. Thus they need to implicitly use both basic FORTH words and periphersal driver words, but want to only explicitly know words that achieve apecific aims. But even here FORTH can be appreciated. It allows a flexible, conceptual system with a nonconfining syntax. With the pulmonary microcomputer, the physician might typically have the following dialog:

\section*{OR PULMONAEY CAICULATIONS}
( acquire data, and calc it)
OR PRINTER SEON RESOLTS
( print resulte)

\section*{OR DME VIEN}
( view plots of gases m) OR C2H2 VIEN

\author{
(... graphics disolav)
}

By leaming a limited, yet full, vocabulary of perheps twenty to fifty well chosen words, these non-technical users can effectively use a FORTH based microcomputer with little training or understanding of programming. And without fail, they learn to use colon definitions to group these basic words to their own specific usage patterns.

\section*{Cormmen Software Packeges}

As we have just seen, I group FORTH software in three coarse categories corresponding to types of users: basic FORTH system software, peripheral support extentions, and custom applications. The basic system software does not vary at all while custom application software is unique to each end-user system. Peripheral support software is in a hazy area. From the point of view of documentation and aupport, any given type of peripheral should appear uniform between systems; but at the hardware level, each type of peripheral varies in myriad details. By creating common software packeges with this in mind I have been able to avoid constently recreating software because of hardwere variations.

Common software packages can do more than just ease support for similar systems. It con effectively hide hardware details from the user, thus making dissimilar \(A / D\) converters, for example. sppear identicel from the software point of view. And a well designed set of driver software aleo imparts increased capabilities to aystem than just those of the "raw" hardware. Let's look at a few examples of software peripheral drivers to reinforce these points.

Many of these microcomputers are used for data acquisition purposes involving different types of A/D converters and real time clocks. From a hardware point of view, some of theae \(A / D\) 's have sight bit versus twelve bit resolutions. Some have seven or eight analog multiplexer channels while others have sixteen. Some of the real time clocks have fixed 60 Hz resolutions, others are programmable.

From conceptual point of view, these data acquisition systems all operate identically: they can randomly sample multiple analog signals at some specified rate. The driver software implements these concepts using two words: SAMPLE and DELAY. SAMPLE takes an integer multiplexer channel number as an input argument, and returns an integer amplitude value. It works identically no matter what hardware is controlled by it; the multiplexer addressing and \(A / D\) digital
sutput format are hidden from the user. similarly, the real time clock works in a manner transparent to hardware wecifics. DELAY requires only an input 3:gument to specify the number of real : ne clock "ticks" to delay.

But the conceptual basis of the data scauisition package transcends just the \(\therefore\) 'D hardware; there must be some place \(:=\) put the data. This may be on the paraneter stack, in data arrays, or in disk sased virtual arrays. When this capability s added, the data acquisition apecific -ardware creates a synergy with the fundanental system hardware such as read/ -ite memory or floppy disk.

Another example of a peripheral driver zackage that 1 developed is a memory--apped viaeo graphics package. The :-Dical hardware interfaces ranged from \(\therefore 0 \times 256\) resolution up to \(512 \times 480\) resolu: \(n\), with as many different methods of sudressing specific dots on the display.

Conceptually, we want, first of all, to \(=e\) able to plot physical \(X, Y\) points indezendent of hardware specifics. A word exch as PLOT, using \(X\) and \(Y\) integer para--eters on the stack top, can give us this zapability very readily.

But to really use graphics effectively, : is nice to be able to specify different areas on the video screen to plot different zata, as well as scaling functions to adopt zgical coordinates to this specified ;-aphics area. The GRAPH data type zullt with a defining word) allows these - fferent graphics areas and scaling func-
 ionmon name. Further capabilities were sided to allow easy creation of vectors, ;-ids, tick marks, axes, and baxes. All of sudden, a very proletarian graphics perizeeral is transformed into a powerful \(: כ \partial 1\). And because these new functions are घ. built on the PLOT word, they are -eadily tansferred between systems with : fferent hardware interfaces.

A final software driver to consider is :-at of the hardware floating point unit. \(\therefore\) is interesting to consider this from both : FORTH, and a conventional language zoint of view. In a language such as \(=\) ASCAL, the system generally has built in \(s=\) ftware based operators for floating zoint. Because the system is not inherentextensible, the addition of a hardware '. sating point peripheral requires either a -anufacturer rewrite of the PASCAL T.Jating point routines, or else a user -terface through PASCAL functions or zrocedures. The former requires manu; seturer acceptance and support of a new -ardware peripheral; unless a very popular sevice, such support will be reluctant at zest. The latter requires a very awkward anguage syntax to invoke hardware float\({ }^{9} 9\) point capabilities. Either way, the
problem is that the hardware has to be forced to conform to the manufacturer's language standard.

At the Medical Center, hardware floating point package was easily added te an extention to the basic FORTH system; the language adopted the hardwara!

\section*{Anmehroniom or Pertent?}

At this juncture it is valid to ask it FORTH justified itself in its use at the University of Rochester Medicel Center. Is it an anachronism of the past, or a philosophy portending the future?

Admittedly, FORTH is somewhat limited without such things as file system or procedural name scoping of variables. Perhaps there should also be less explicit knowledge of addresses, and more system serurity. Perhaps. But if so, then these things will be evolved as FORTH matures.

It is what FORTH espouses, though, that justifies its use. It allows hardware components to dictate the software design, thus allowing rapid incorporation of technological advances. Other languages force conformance of hardware to language standards--a slow, expensive process.

FORTH allows isolation of users from hardware dependencies, and adds capabilities to the basic hardware. The result is a user environment that supersedes specific machine configurations with concept oriented, yet free syntax, computer operation. The FORTH system developer might need to know "how", but the system user need only know "what". Conventional systems, to the contrary, generally require everyone concerned to ask: "why?"

FORTH encourages an exploratory development technique. A user can choose between interactively trying concepts, writing full programs, editing programs. compiling programs, and/or debugging programs. He or she can do this in a single, consistent FORTH environment, utilizing any of these phases of development as required. The result is efficient use of all system resources.

The embodiment of the FORTH phllosophy is that programming is not what it is often taught to be: the application of topdown programming techniques to a single problem. Instead, it involves a series of interrelated problems all related to system use. This might mean a set of words that allow a researcher to control a TV digitizer, or it may mean eseries of words to calculate and graphically display the results of a mathematical analysis. While the apries of capabilities needed will always vary between different syatems, it is only by providing a rich enouah vocabu-
lary that a user con have a flexible, effective, and friendly system. FORTH is unique among languages in that it encourages the programming of solutions!

Peter Helmers is a senior laboratory engineer in the diagnostic ultrasound research laboratory within the Department of Radiology at the University of Rochester Medical Center.

\section*{Eelvere' article continued on next two pages}

\section*{BUG FIXES}

\section*{Correction to FEDIT}

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

\section*{1. SCR 64 Line 10: compile should be COMPILE}
2. SCR 65 Line 23: 1+/MOD should be 1+ \(16 / \mathrm{MOD}\)
3. SCR 67 Line 48: B/BUD should be B/BUF
4. SCR 67 Line 49: : E should be: .E
5. SCR 67 Line SO: + 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 in FD 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 figFOR TH line editor definition, \(R\). I used the pseudonym because FEDIT was the first program 1 wrote in FORTH and I wasn't really farniliar enough with Vocabularies to comfortably use a word that was already uaed in the FORTH vocabulary.

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

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


Figs 1: Elack diagram of a typical S-100 baed mierocomputery this one in uned to atedy blood vein mechenice.



 yotern. Baceuve the microcomputer cen oynotronize to timbug heroware, ditur oppediItioi euch a extemintior and froqumey control can be etillized.


Fig. 3: Diegram of vein mechenice experimental chenber. Microcomputer mornplese previere and force aigneles, and determines vein dierneter from softwere menalyate of TV Image.


Fig, 4s Dingrem of X-ray scarner appmetue showing how wheel collimetor and fore and aft harizontal collimators, controlled by stopper motore, create a mechanlcelly scenried \(X-r a y\) reeter. The microcomputior, with \(N O\) and D/A interfecee, atioo monitore and controle \(X\)-ray expoeures.

\title{
DATA STRUCTURES NA \\ TELECOMMMNCATIONS FRONT END
}

\author{
John A. Lefor \\ University of Rochester
}

\section*{Asbtract}

URTH, the University of Rochester dialect of FORTH, was used to implement a telecommunications front end for an IBM 3032. This package provides access to the IBM 3032 from as many as 160 ASCII terminal at speeds up to 9.6 Kb . Each of these terminals contend for 128 simultaneous connections at the IBM computer.

The reasons for choosing URTH as the development language and a review of the major advantages and disadvantages of using Urth for this project is discussed. Also, some conclusions as to the applicability of URTH, and the data structures used in this application is reviewed. The use of conventional data structures for providing information paths between the various components of the system is examined and the possible advantage of less conventional data structures more firmly based in URTH constructs is explored.

A plan for development of similar systems is presented which integrates some of these concerns and promises a better structured system.

\section*{Introduction}

In 1977, the University of Rochester Computing Center first got involved with the FORTH language. The initial development in FORTH was the implementation of various flavors of the FORTH system known collectively as URTH. Most of the URTH systems developed have provided multitasking capability on a variety of micro-, mini-, and mainframe computers. During the development of the various URTH systems, a number of people within the Computing Center showed interest in using an URTH based system for development of real projects rather than viewing URTH as just another academic curiosity.

Concurrent with the development of the URTH system, was the growth of telecommunications in computing at the University. A need for additional telecommunications lines into the computer was fast becoming a necessity and the financial support for such a purchase was on the verge of becoming a reality.

In this environment, the design and implementation of a locally deaigned telecommunications front end was beginning to emerge. The front end had to exist in an academic computing center where the need for teleprocessing was growing. The front end had to communicate with an IBM host (it was generally believed that the

IBM environment was at the University for many years to come). The front end had to provide access for the ever growing number of ASCII terminals being purchased for both computing and noncomputing environments. Importantly, the front end had to provide for access to the IBM host from more terminals than could be dedicated to the host at any one time. The only front end which could posaibly meet these goals and be reasonsbly cost effective had to be one of local design. meeting local requirements.
Features Provided
The front end designed at the University of Rochester Computing Center does provide some unique features to the users of our IBM 3032 computer. To be sure, the features are not unique within the context of computing, but are not generally available in an IBM mainframe environment.

One of the major advantages provided by the locally designed front end is the ability to switch between systems from the same terminal. In a traditional (nonSNA) IBM mainframe, it is not always convenient to have a terminal switched between different software teleprocessing applications. Typically, a terminal either is connected to one application or another. With the locally designed front end, it is possible to choose the application ot which the terminal is attached. In effect, the front end is a port contender for various applications on the mainframe.

The second major feature arising from a local front end is the ability to support an XON/XOFF protocol. Since the IBM mainframe communicates with its terminals in a half duplex mode, XON/XOFF support is not traditionally available. The local front end is based on full duplex communication to the terminal 80 XON/XOFF can be supported in a fully effective fashion. Those terminals which have buffers which can overflow can turn off the input at will, a feature not available without special support in the IBM world.

The front end is today rumning at the University of Rochester Computing Center. It is supporting 160 ASCII terminals contending for 128 host computer ports. Each terminal can select connection speed between 110 and 9600 Baud as well as a few other tailored features. The fact that the implementation continues to run frequently eppears to be a miracle but represents some faith that the concept is at least essentially mound.

\section*{Hardware Deciaione}

In order to implement the telecommunications front end to an lBM computer, the procemsor chosen for the implementation had to provide the capability to interface to an IBM byte multi-
plexor channel. Since the protocol for channel interfacing is non trivial, there are a limited number of vendors of minicomputers who were able to provide this interface capability. Another important consideration in the design of a telecommunications front end is the realization that if a failure should occur in the front end, there is a perception that the host computer failed. Because there is great need to access the host computer, it is undesirable to have hardware failures sffecting the front end. To this end, the mini-computer chosen as the froat end had to have both a history of reliable service and a maintenance teaiti capable of repairing any difficulty with a minimum of fuss.

In evaluating the available minicomputers against these criteria, the processor which was finally chosen was a Digital Equipment Corporation PDP 11/34. The interface to the channel is via a DX-11B, and the ASCII terminals are supported by DZ-11's (actuaily many of the terminals are supported by a Digital Communications Associates 205, which emulates 32 lines of DZ-11 on a singie quad height board).

In retrospect, we can see that though the PDP 11/34 does work in the required environment there are some deficiencies. The moat notable is in the maintainability of the \(D \times-118\) (the channe! interface which connects the PDP 11/34 processor to the IBM processor). There are so few DX-11B's in production throughout the United States that the DEC customer engineers are relatively unfamiliar with the details of its operstion. When subtle problems have occurred, the repair of the problems has taken considerable time and talent. To be sure that the subtle difficulties were discovered and corrected is a tribute to the engineers dedication to the problem, but a more popular interface would probably have been repaired in a shorter time.

\section*{Software Daciaiona}

In determining the nature of the software to run for this application, it was necesaary to evaluate the probable structure of the end goal and to consider all the concerns of a project of this sort. After the major considerations are evaluated, the best software choice can be made based on the concerns and knowledge of what is available.

A telecommunications front end is a realtime device which must be able to handle a relatively large number of potential \(1 / O\) devices. In particular. many terminals are expected to be connected to the front end. Also, there were considerations for attachment of synchronous lines for support of Hasp Bisync, Remote 3170 's, and local area network communications. All these considered together, it
was important to choose a software .nplementation which provides support for -eltime device handling.

The wide variety of \(1 / O\) devices which were contemplated for the front end also -evired that the software provide tools to relp the designers of the system gain understanding of a wide variety of hardware devices. There were going to be asynchronous and synchronous devices as well as a channe! interface which had no well defined characteriatics (the best socumentation of how the DX-11B worked was found in the diagnostic programs supz:ied for hardware maintenance). In addition, there was always the possibility \(z^{\prime}\) needing to support a new and different siass of I/O device. Though the manuals socumented how the hardware worked, 3ny software which would allow interastion with the unfamiliar hardware would ze beneficial in the debugging of the overa.: system.

Another area of debugging which was :onsidered in the software choice was the sifiware protocols. The connection to the :nannel of an IBM computer by asyn:nronous ASCII devices invokes a non-\(\because\)-ivial set of software protocols. A simple Example of the kinds of problems is in the :-ansmission of any single ASCII character \(: 2\) the channel. In the IBM environment, :e software running in the processor expects that any ASCII characters trans--itted from a telecommunications front :nd are sent not as simple ASCII : naracters (as generated by the terminal), \(=-\) : rather demands that each ASCII charsater be bit reversed. Though this is not ; difficult feat to accomplish, it points xit the nature of some of the software :-otocol issues which must be dealt with a telecommunications front end. Esffice it to say the software used to -esign the front end would benefit the eesigner if it helped to identify, and \(\cdot \mathrm{e}\) solve, software protocol issues.

In the development of any realtime software project, it is recognized that the :-roughput of the system is important. he telecommunications front end is no exception. Since there are to be a large --mber of \(1 / O\) devices providing input to :-e software application asynchronous to : e operation of the software, it is impera:ive that the application software be able \(::\) keep pace with the demand. On the :: her hand, the inability of the front end \(::\) keep pace with the demand is not criti:3.. If a character destined to a terminal s lost, a human being will not die but a ::ogrammer may get upset. Keeping : ese priorities in mind the project had to =e implemented in an environment which -as not wasteful of processor time, but :here was no need to be alarmed if there -as the potential to loose data.

The hardware decision made specific 'eatures of the processor had to be con-
sidered in the software choice. Spectfically, the PDP \(11 / 34\) had 64 K bytes of memary. We had to have some degree of confidence that the entire system could be packaged in 64 K bytes. If that was not possible, the development time could be slowed down waiting for shipment of additional memory. The spaed of the 11/34 processor led us to believe we would have sufficient CPU to do the job, but not a lot to spare.

The final and perhaps major consideration which affected the choice of software was the perceived development time. The project was initiated at a time when there was an extra IBM processor at the University. It would be possible to design and debug the entire front and on a processor which was not in use. That was a real opportunity not to be passed up. However, the processar could not remain idle for too long a time. Any software package which could help to shorten the development time and thereby allow debugging of the front end on the unused processor would be of great benefit to the implementation.

\section*{Alternative Software Strategiee}

Examining the issues in making the software choice, there appear to be three alternative software strategies. The use of assembler language, the use of a high level language such as \(C\) or Fortran, or the use of URTH.

Assembler language provides a number of solutions to the problems outtined. It tends to be compact in mernory usage, it certainly has the potential to make most efficient use of the limited CPU, and it is quite capable of handling the foreign devices needed for a front end. However, the assembler has a few drawbacks. Probably the major difficulty with assembly language is the extended development time. Debugging is slow and tedious and design of code and data structures to aid debugging is totally a responsibility of the programmer. Thus. development of a major application in assembly language is concerned both with the solution of the problem but also much effort is spent on good design and coding techniques. Another difficulty with the assembler is maintainability. Each programmer has an individual design style. The documentation rests largely in design of the code. If the original designer is no longer available for maintenance of the project, there is a long learniny curve to train new indi vidual.

High level languages solve many of the difficulties with assembly language. If the language is well conceived for a realtime problem, it will support the difficult hardware issues and will provide a framework for data structure design which provides readability and maintainability of the software. A major difficulty with high
level languages is their use of memory, and sophiaticated operating system services. These two concerns may make a larger faster CPU needed for effective execution of the application. Another drawback of both the assembler and high level solution is the lack of inherent interactive develoment and debugging tools. They typically can be designed into the system, but they generally are not present in the basic environment.

\section*{Evaluation of URTH}

URTH appears to meet many of the goais in the software choice. Though there are limitations, the advantages seem to outweigh the disadvantages especially when design time is so important a consideration.

When looking at URTH, a clear advantage fiforded by URTH is implementation time. 2 Most of the other advantages provided by URTH can be directly tied to the speed of implementation, URTH provides easy eccess to any set of unusual devices, because the device handlers are ach tailored to the system and the hardware. Once a program is debugged in URTH, there is good reason to believe it will continue to work. \({ }^{3}\) Another major advantage offered in the URTH environment is the enormous flexibility in design of both source codes and data structures. The ability to code both high level URTH and machine level code and to achieve a uniform interface provided many opportunities to speed up inefficient code. The ability to design new data strucutres to work in a large scale environment offers much flexibility in design.

The URTH environment is not without feult. The fact that URTH is an interpreter does mean the code is not as efficient in CPU speed as possible. Of course, the ease of generating assembly code helps alleviate this problem. However, a major drawback of the URTH onvironment stems from its flexibility in data structure design.

The very fact that it is possible to design any needed data structures coupled with the implementation of the traditional data structures of arrays, constants, and variables created some difficulties in the design of system which had so much pressure for development in a short time. There was not a lot of time spent on development of the best data structure for the problems encountered. Rather, traditional data structures were used to meet individual demands. In particular, many arrays were implemented for storing of information relating to specific \(1 / 0\) devices, and queues (obtained from a freepool) were used to buffer data between devices. The use of such data structures had two major impacts on the project. First, the queues were sufficiently difficult to handle as to have impact on the
speed of the overall system. \({ }^{4}\) The use of the arrays to hold information for later processing yielded much difficulty in debugging individual words and tended to leave side effects which had impact on words already debugged.

Thus, the use of URTH has many virtues but it is crucial to recognize the particular issues which may lead to difficulty in debugging. Using data structures such as arrays and variables to communicate information between tasks in the front end tended to leave open many portential pitfalls in the debugging and design of a system as complex and highly integrated as a front end.

\section*{Alternative Deaign Strategies}

In examining the resulting front end for difficiencies, it becomes clear that there are some strategies for alternative design which could limit the difficulties encountered in any similar realtime project, and would make URTH a vehicle for well designed, well integrated, and effective systems design.

The issues of code design are well considered in URTH. The ability to switch between machine level code and high level URTH provides a classic tradeoff between speed of execution and memory utilization. The fact that the interface between both environments is standard allows all design in high level URTH, and conversion to machine code when and where appropriate. In this area, URTH provides suffficent tools and a good set of options.

In the data design area, URTH provides so many options that the best data structure choice is very much at the control of the programmer. In the case of the front end design, the traditional data structures were not sufficient to effect the job but there was insufficient time to design an optimal data structure. In retrospect, it is possible to peruse the alternatives and choose a structure which provided the flexibility needed, and also limits the side effects from preventing effective debugging of words.

One of the major advantages URTH provides over alternative software approaches is the stack. Proper design of URTH words with parameter passing via the stack helps to insure that a debugged word will tend to continue to work, and will have no side effects Given this observation, it would be natural to use the stack to pass parameters in the telecommunications environment. Unfortunately, the stack is not useful in communication between tasks, and the stack is difficult to address and use when too much information is passed. In the front end, there are \(s 0\) many unrelated parameters which need to be passed between tasks that the stack is not useful. But, the concept of a stack
does solve one of the major difficulties encountered in the front end design. Given this set of considerations, it seems like a good jdes to define a named object stack \({ }^{55}\) for each I/O entity defined in the telecommunication environment. When a particular 1/O device needs some form of service, the named stack is invoked and all data relating to the 1/O device is available. The stack can contain pointers to ring buffers as well as current status of the device. Using this strategy providas an environment that naturally fits within the basic strucutre of URTH programming, makes effective use of constructs within the URTH system, and promotes good URTH programming practices which minimize the side effect problems. Overall speed of the application is not significantly impacted and meny ald functions can take advantage of the data structure.

The stack will contain sufficiont volumes of information about each \(1 / 0\) device that it may be advisable to create a "framing" of the stack. This would allow access to individual parts of the stack as if it were the current top of stack, thus allowing access to more data in aconvenient notation.

\section*{Surmmery}

The telecommunications front end designed and implemented at the University of Rochester Computing Center is a useful model of many realtime applications. In the design are found a number of flaws which are primarily related to the particular pressures present at the time of the design. The choice of URTH as the software vehicle appears to have been an excellent one however, the choice of data structures to use within the URTH environment was not as well conceived.

URTH provided a software environment which clearly effected time effective development of complex aystern. It provided a comprehenaive interactive debugging environment with the ability to address apecific speed inefficiences in aniform manner. The major drawbacks to the URTH environment resulted from the choice of data structures for intertack communication within the application.

URTH does provide toole to develop the optimal data structures for any particular application. In the case of realtime applicationt, the choice of data structures is particularly critical. From my experience, 1 believe that a data structure similar to the -named object stack would benefit many realtime applications in URTH both function provided and in the limiting of side effects 30 prevelant in global data strucutres such as arrays.

A second feature which would be valuable in URTH environment would be
any useful stand-alone dump with indexing to help the programmer walk through the dictionary. When total application collapse occurs, URTH is not very informative as to the nature of the problem. A memory dump (with a good index for the dictionary) would help to debug some rather sticky timing problems.

Overall, URTH is a good choice for development of reeltime applications, but care in the design of data structures should thelp to make the overall mainte. nance of the application a simpler chore.

\section*{Footncter}
1. This is not simple an example of a perverse IBM, but instead is another fact of IBM computing history. The standard device IBM used to connect ASCII terminals to the host ( \(\mathbf{a} 270 \mathrm{x}\) ) was not designed using today's UARTS, rather it collected the bit serial data in a register. The data was collected in a register in such a way as to cause the charecters to be captured in bit reverse order. Rather than correcting the problem in the front end, they tranamitted the bit reversed ASCll to the hoat, and translated the bit reversed ASCII to EBCDIC for processing. The software stayed, so the need for bit reversed ASCII exists today.
2. This adventage was certainly realized in the actual project. The basic system wes operational within four months from beginning of the project.
3. This is dependent upon good URTH programming practices. But, in our project there became clear a self evident truth. We attempted to debug \(s 0\) many "words" which were already correct, we began to believe that it is very difficult to debug a working program.
4. Converting most of the queues to individually assigned ring buffers speeded up overall processing by \(20 \%\) or more.
5. See Peter Helmers, "Userstack", FORTH DIMENSIONS, Vol. III, No. 1 and Peter Helmers, "Alternative Parameter Stacks", Proceedings of the 1981 Rochester FORTH Standards Conference.

\section*{Acknowiedgements}

I would like to thank Richard Marisa, Ken Hardwick, Mike Armstrong, and Mike Williams for their asaistance.
J.A. Lefor was senior systems programmer at the University Computing Center at the University of Rochester and is now Data Communications Manager.

\title{
MAPPED MEMORY MANAGEMENT TECHNGUES IN FORTH
}

\author{
Rosemary C. Leary \\ Carole A. Winkler \\ Laboratory for Laser Energetics University of Rochester
}

Abstract
Three techniques for using memory managernent hardware in a FORTH system ave been implemented at the Laboratory for Laser Energetics at the University of Rochester. One method uses mapped 7emory for data storage by creating a "Jata window" in the logice! address wace. A second method increases the available space for programs by mapping :asks in a multi-tasking system. The third -ses mapped memory for data storage by :axing sdvantage of special instructions and a second set of memory management -gisters.

\section*{introduction}

The problem of insufficient memory ar programs or data is commonly encouniered on computers with a 16 bit word ;ze. Many manufacturers now offer hard--are to alleviate this probiem. At the -niversity of Rochester's Laboratory for -aser Energetics we have devised solu\(\therefore\) ans to three different aspects of the \(=-\) oblem using FORTH on PDP-11/23 and \(=\sum^{0}-11 / 34\) computers.

Two applications at the Laboratory had三 need for large image processing arrays s to look words). We solved this by - Eng a double precision array index which -aps physical memory into a logical mem\(\because\) :- "data window" within the FORTH sys: 7.

On a different, very large FORTH ap: cation, we needed both more program cace and more data space. We increased :e amount of program space by imple--enting a multi-tasking system in which :ertain portions of memory contain the \(-\quad\) - leus and common code, while other :z:: ons are task specific and are period: \(3 l l y\) switched in and out of active use.

To increase the available data space -e are using special instructions and a sezond set of memory management regis--es on the PDP-11/23 and PDP-11/34 : :mputers.

Additional material on these systems : an be found in "FORTH in Laser Fusion," :- Larry Forsley, in this issue of FORTH こ:MENSIONS.

\section*{tardware}

The memory management hardware on ve PDP-11/23 and PDP-11/34 computers : nnsists of two sets of registers that map \(\therefore\) bit logical addresses into 18 bit phys-
ical addrasces. One set of registers is used when the processor is in "kernel" mode, the other when it is in "user" mode. The mode is determined by two bits of the proceseor status word.

Each set of registers contains eight 32bit Active Page Registers (APR's). Each APR is actually two registerss the Page Address Register (PAR) which containe a base address, and the Page Deseriptor Regiater (PDR) which contains the page length and the access control key.

The 16-bit logical address epace is divided into eight "pages" shown in
Table 1. When the mamory management unit is enabled, any access to memory will be mapped through the APR for that address.
\begin{tabular}{cc} 
Fase & Losicel Address Ranse \\
(octal)
\end{tabular}

Table 1. Losical Address Space.
The physical memory address that will actually be accessed is a combination of the logical addrese and the PAR for that page. Figure 1 shows how the logical address is deriv: \(d\). Bits \(15-13\) of the logical address give the page (or APR) number. The PAR for that page gives the base address in 64 byte blocks. This value is added to the block number field of the logical address (bits 12-6) to find bits 17-6 of the physical eddress. Bits \(5-0\) of the physical address are the same es bits 5-0 of the logical address.

Figure 2 shows the logical address epace.


Figure 2. Logical address space for single task without mapped memory.
Additional information on the PDP-11 memory management unit can be found in the processor handbook․

\section*{Deta Window and Mernory Managernent}

One way to utilize the memory managament hardware and additional memory is to use it for data storage. Two of our applicetions at LLE require large data errays (up to 100 K words) for image processing. We solved this problem by cresting a "data window" in our logical address space. Figure 3 shows the logical address layout of a system with a data window.


Figure 1. Construction of a Physical Address
(derived from figure 7-9 of \([1]\) and
reprinted with permission from DEC.)

Figure 3.


Logical Address Space With Data Window.
The block buffers, return stack, and parameter stack are moved down to the top of the next 4 K word page of logical memory, leaving a \(4 K\) word gap in the logical address space. In a 128 K word sygtem, 100K words of physical memory are then accessed through this window.

The \(X\) and \(Y\) coordinates of the image array are converted to a double precision index. This is done by multiplying the \(Y\) coordinate by the number of pixels per line and adding the \(X\) coordinate. This index is divided by the number of pages per image. The quotient indicates which page the pixel is in, and the remainder will be the address offset of the pixal into the page.

The relocation constant for the needed page is set in the PAR so that it cen be accessed through the data window. The logical address of the pixel is obtained by adding the addrees offeet to the starting addrese of the deta window.

\section*{Multi-tadding and Mernory Menagoment}

Our version of FORTH implements multi-tasking in the following manner. Each task has a "atate vector" which contains "user" variables that can differ from task to task. This includes:
- Dictionary and atack pointers
- Program counter and interprater pointer
- Status flags and state indicators
- Terminal I/O routines and buffer pointers
- Vocabulary pointers
- Number base

The state vector for the master tack in included in the nucleus.

Each task also has ite own terminal buffer, dictionary, parameter stack, and return stack. New tasks are created with a routine called BLDTASK which allocates
space for them in the master task's dictionary. Figure 4 showe the logical address epace in en unmapped multitasking system.


Task atate vectors are linked to each other in a circular fachion, one pointing to the next and the last back to the first. A "round robin" scheduler starts running new tack when the current tack executes a PAUSE. PAUSE etores the current machine state into the state vector of the existing taok and eete the new machine state according to the new takk's state vector.

Additional information on multitasking \(c_{3}\) n be found in works by Forsley \({ }_{4}^{2}\), McCourt \({ }^{3}\), and Leary and McClimene \({ }^{4}\). Figure 2 shows the logical addrese apece of a FORTH application with a eingle task and not using memary menegement.

To add program apace to our multitaaking aystem, we reserved atank windown in the logical address apace. The master task occupies the low five pages of address apece. Code in this area is usemble by all taske.

Mapped tacks occupy pages 5 and 6 of the logical address apece. Definitione and data within a mapped task are acceasibie only to iteelf. Each task muat have 8 eeparate vocabulary. If definitions in a mapped task are entered into the FORTH vocabulary, the dictionary lirks will be gane when the next task becomes active. This usually results in system crash. Figure 5 shows the logical sddrese apece in a mepped multi-tasking system.


Figure 5. Logical Address Space for Mapped Multi-tasking System.

Implementing this technique required the following changes:
- Modify the scheduler PAUSE so that it sets the page 5 and 6 memory management registers, as well es swapping in the usual state vecter information.
- Move the block buffers and master tadk stacks to the top of page 4.
- Chenge the routine BLDTASK to asaign the new task's return stack, parameter stack, and dictionary to peges 5 and 6 , instead of giving them space in the master task's dictionary.
- Change BLDTASK to assign physicel memory to the task. It must calculate the appropriate settings for APR 5 and APR 6 and save them in the task's state vector so that they can be loaded into the memory management registers by PAUSE.

\section*{Her Space for Deta}

The two epproaches discussed previously both ran in procesaor "kernel" mode. To increase our memory resident data storage in the multi-tasking application described previously, we use the "user" mode memory management registers.

The proceseor status word has two mode fieldas current mode and previous mode. The instruction MFPD moves a word from the "previous" mode address epace to the "current" mode processor steck (the return stack in our FORTH implementation). The instruction MTPD moves a word from the "current" mode procestor etack to the "previous" mode edtress apace.

Using these instructions it is possible to retrieve and store data quickly and
at．siently，and the data stored there is zasessible to all kernel mode programs， －－ether they are mapped tasks or not． －a：a tables that would otherwise need to ：？Jisk resident because of their size can －＊be memory resident to speed response ＂e．

The source listing of the user mode \(: \equiv\) ：a storage code is included at the end of －s article．

\section*{Encluaion}

The first technique，the data window， －is been used for two image processing ：zz．lcations．One is used to view infrared ミ－－ultraviolet laser bearns in materials ：：－age testing experiments．The system ：ues circular averaging and calculates an Esolute intensity within the 10 minute \(-:\) ：cycle．

The other image processing application ：こse－ves \(X\)－ray diffraction patterns pro－ Fied by a nanosecond X－ray source．A \(\because=\) nique of radial averaging is also used \(\because-\)－to enhance the diffraction pattern ＝－：study changes induced by sample stim－ シャ．วn．

The second and third techniques are \(\because=\) on the Omega Alignment System， －－ h now has 17 tasks installed and uses I2：－140，000 bytes of memory for pro－ ＝\(\equiv \mathrm{m}\) space．The user mode data storage －\(=:\) hod is used by the data base software \(z^{-}\)－－or the intertask message queues．

Aithough this paper describes tech－ ：ies used with DEC PDP－11 series com－ \(\therefore\). ．e：s，the techniques are similar to those \(\therefore: \therefore\) with any limited address system with ：：：al／physical mapping hardware．Thus， －e．are applicable to minicomputers like －Hewlett－Packard 1000 series and the －． n newer 16 bit microcomputers like \(\because\) Motorola 68000 and Zilog 8000．The \(\because:\) niques are especially appropriate in a －TH－79 context where the FORTH －F－7ine is defined as having a 64K byte m－：－ess space，carved out of an arbitrarily \(\underset{\sim}{\tau}\) physical address space．

\section*{Lexnowledgements}

Te following people played a major －\(\equiv\) in the development of the software zis：－bed in this article：Donald P． －Z．imans，Lawrence P．Forsley，Reade －vmick，Robert D．Frankel，Joseph A． －：：：e，and Robert L．Keck．
－his work was partially supported by －＇Jllowing sponsors：Exxon Research I：Engineering Company，General Elec－ ＂：Company，New York State Energy ＂esearch and Development Authority， －．：east Utilities，The Standard Oil ：－－pany（Ohio），the University of ＝－mester，Empire State Electric Energy ＝seearch Corporation，and the U．S． －astment of Energy inertial fusion －G：am under contract number DE－AC08－玉＝ 20124 ．


\section*{\(i 5\)}

R．C．Leary is a consultant employed by the Engineering Division of the Laboratory for Laser Energetics．C．A．Winkler is an undergraduate in the Department of Mathematics，University of Rocheater．

\section*{References}

1．Microcomputers and Memories，Digital Equipment Corporation，Maynard，MA 01754， 1981.
2．Lawrence P．Forsley，＂FORTH Multi－ tasking in URTH，＂Proceedings of the 4th West Cosst Computer Faire， 1979.
3．Michael A．McCourt，PDP－11 FORTH 79 Implementation Guide，University of Rochester，Laboratory for Laser Energetics， 250 East River Road， Rochester，NY 14623， 1981.

4．Rosemary C．Leary and Donald P． McClimans，Omega Alignment System Software Maintenance Manual，Univer－ sity of Rochester，Laboratory for Laser Energetics， 250 East River Road， Rochester，NY 14623， 1981.

\section*{FORTH CLASSES}

November 16－20
Decernber 7－11
January 11－15
Call：Inner Access
（415）591－8295

\title{
A HGG LEVEL NTERRUPT HANDLER N FORTH
}

\author{
R. L. Keck and L. P. Forsley \\ Laboratory for Laser Energetics \\ Unversity of Rochester
}

\section*{Abstract}

A system for writing interrupt service routines in high level FORTH is described. An example of the utility of high level interrupt service in a dynamic data acquisition situation is provided.

\section*{Introduction}

X-ray data from laser-plasma interaction experiments on the GOL laser system at LLE has in the past been acquired from photographs of oscilloscope traces. Because of the large number of detectors currently being employed, this method has become impractical and we have chosen to use 12 channel integrating A/D converters for data acquisition. These A/D converters are CAMAC \({ }^{1}\) compatible modules and because of the extensive CAMAC vocabulary available in the UR FORTH-79 system, as well as the suitability of FORTH for use in a dynamic programming environment, FORTH is used for the acquisition software.

The \(A / D\) modules integrate the signal at each of their 12 inputs for the duration of a gate signal, which is derived from the laser oscillator. The oscillator is fired once every 10 seconds to keep it in stable operation, however, our data signal occurs only when the full system of laser amplifiers is fired as well, an event which occurs when a fire sequence is carried out by the laser system controller on command from the operator. We require a means of clearing the \(A / D\) modules just in advance of the oscillator pulse at which the full system will fire. This is accomplished by feeding a ready-to-fire signal, provided by the laser system controller 4 seconds in advance of fire-time, to a CAMAC contact sense input module. Our acquisition sequence then is: look for a ready-to-fire signal from the contact sense input module, clear the A/D module, wait for data available indication from the A/D module end read the data from the A/D module.

The above sequance could be implemented directly, using the available CAMAC vocabulary, by simply continuously interrogating a module until the desired condition occurs and then proceeding to the next step. This method needlessly ties up the computer executing loops and prevents it from handling any other task while the sequence is in progress. Since both the contact sense imput module and A/O module will generate CAMAC Look At Me's (LAM's) when a aignal occurs at their imputs and a CAMAC LAM can generate an interrupt, we can
use an interrupt driven acquisition system which will avoid needless looping. This requires the writing of interrupt service routines in machine code, which is at best cumbersome. It would be nice to be able to write high level FORTH interrupt service routines which could be readily changed. This cen, in fact, be done and our method for doing this is discussed below.

\section*{Implementation}

Our system consists of UR FORTH-79 ruming on a Digital Equipment Corporation LSI-1l microcomputer under DEC's RT-11 operating system. While a complete description of the implementation of this system may be found in the implementation guide \({ }^{2}\), we will briefly cover FORTH's usage of processor registers for reference in the following discussion.

Four of the processor's general purpose registers are dedicated FORTH registers. R6, the system stack pointer, serves as FORTH's return stack pointer (RP). R5 is used as the stack pointer (S). R4 is used as the FORTH interpreter pointer (IC); it contains the address of the compilation address (also referred to as the code field address or CFA) of the next word to be executed. Finally, R1 is the state vector pointer ( SV ); more will be said about the SV later.

The procedure for executing a FORTH word from code is essentially quite simple and is accomplished by the word XEQ.MACRO (a listing is included in the appendix). It accepts an address, into which will later be placed the compilation address of the interrupt service word, on the stack and generates code which will place the compilation address of the service word on the stack [MOV @ \(\angle A D D R\) ,-(S) ], loads the IC with the address of the compilation address of the return from interrupt code [MOV \#AHERE+8zIC ] (note that <HERE +8 > contains the compilation address of RTI (COMPILE RTI), the return from interrupt code word) and then jump to the executable code for EXECUTE to begin execution of the interrupt service word [JMP ' EXECUTE]. The net effect of this code sequence is to start execution of a high level interrupt service word and subsequently execute the return from interrupt code.

Before execution of the code generated by XEQ.MACRO can begin, the contents of the processor registers must be preserved by pushing them onto the system stack. Code to do this is generated by REG.SAVE.MACRO. We must additionally ensure that the 5 and SV registers point to valid memory areas. In the multitasking UR FORTH-79 system, this is most easily accomplished by having a separate interrupt tadk area. The task area contains retum and parameter stack memory allocations as well as a state
vector allocation. The SV register points to the state vector and the state variables contained in the state vector are addressed relative to the value of the \(5 V\) register.

It should be noted that it is not necessary to have a multi-tasking system in order to implement high level interrupt routines. This is because the values of the state variables referenced by the interrupt: routine are in general identical to those for the master task. On a non multitasking system we would simply reserve a parameter stack area for the interrupt routines and set \(S\) to point to it, It is necessary, however, that FORTH be coded reentrantly for this scheme to work.

The SV.SET.MACRD is used to generate code which will set the \(S V\) and 5 registers. Note that it also changes the return stack location. This would not be necessary, except for the fact that the FORTH stack checking routines require that the return stack be located in memory immediately above the parameter stack. The value of the interrupted task's return stack pointer is stored in a free vector location [ \(52 \mathrm{~T}(\mathrm{SV})\) ].

SETUP.INT sets the interrupt vector, in this case specifically for CAMAC (the vector for the device in slot \(N\) for the CAMAC crate is located at \(400+N * 4\) ). The processor is run at priority 7 during interrupt service to prevent further interrupts from occurring.

To make it simple to create interrupt service routines, the macros previously discussed are combined to produce a defining word called

\section*{CREATE.CAMAC. INT.WORD .}

This word when executed, accepts a task area and CAMAC slot number on the stack and creates a word which contains the code sequences previously developed starting at the second parameter field location of the newly created word and sets the interrupt vector to point to this code. The first parameter field location is reserved to hold the compilation address of the word to be executed when an interrupt occurs. The DOES> part of the new word will load this reserved location with the compilation address of the desired interrupt service word.

\section*{An Example}

The listing for blocks 3 and 4 illustrate how the interrupt handler is used in our acquisition systern. A task area (ITASK) is created and initialized for the interrupt routines to use. It must be delinked from the multi-tasking system to make it transparent to the multi-tasking dispatcher. Then two interrupt service routines are defined (RDY.WORD and FIRE.WORD) each with an associated CAMAC slot (or
sevice）．Thay share the same task ares s．nce only one interrupt service routine ：an be active at a time．

In block 4 ，the high level service －astines are defined．RDY．INT is used to ：ear the \(A / D\) module，enable \(A / D\) LAM＇s \(\times\) CLR XENLAM）and then clear and dis－ asie further LAM＇s from the contact senae －ou：module，on occurrence of a LAM ＇：sn the contact sense module．FIRE［ \(=\) ：ilects the \(A / D\) data，disables further \(\therefore\) LAM＇s（XCOLLECT XDISLAM）and a：：：vates another task which will print the －esults（2TASK DISPATCH）on occurrence \(=\) e LAM from the \(A / D\) module．These －gn lave！routines are installed as the －：errupt service routines for the appro－ ：＂ate CAMAC devices with the sequen－ ：es：RDY．WORD RDY．INT and \(\because\) RE．WORD FIREL．Changing an interrudt se－vice routined with this system requires ：－． \(=-0\) and installing it as the hancler word， ：y．，FIRE．WORD FIRE2［ will make the －std FIRE2［ the new interrupt service \(\because\)－ine for the \(A / D\) module．

Enclusione
We have shown that it is possible to －．：e high level interrupt service routines ＝JRTH．This makes it possible for pro－ －－ammers unfamiliar with interrupt pro－ \(\because j\) nming to easily write interrupt service Odines．In addition，the facility with －－oh this system permits changes to be －ade to the interrupt handlers makes this ：－teal way to handle data acquisition in ：：apidly changing experimental environ－ －ent．

\section*{Aknowledgement}

The authors would like to thank －Ehael McCourt for assistance with se：ails on the internal operation of UR ここ？TH－79．
＝．．Keck is a graduate student in Mech－ \＃n＝al Engineering at the University of ＝shester．L．P．Forsley is Group Leader ：－Oomputer Systems at the Laboratory ：－Laser Energetics，University of ＝：こnester．

BLOCK

（ Yish level FORTH interrupt handler rlk lpf 25－mas－81 ）
：REG．RESTORE，MACRO（〈〉－〈〉，REstore Penisters 0－5（） ASSEMHLER 0 S NO 1 RP \(>+\) MOU，-1 ＋LOOP FORTH
CJDE RTI（ restore reasisters，return from interrupt（） RP 52T SU I）MOU，REO．RESTORE．MACRO RTI．FORTH
：XEO．MACRO（＜addr of xea word，essembly time＞－〈〉（） ASSEMELER \(8-\)－）SHAP © MOU，（mush handier word eddr on stack， IC HERE 8 ＋MOU，（ preset the IC ）
－EXECUTE F JMP，（ Jums to execute） COMPILE RTI（ Pointer to next instruction）
FORTH：
 ASSEMBLER 60 DD RP - ）I MOU，LOOP FORTH ：
－－＞

（ more interrupt stuff 25－may－81 rlk）
：SETUF．INT（＜slot＊〉＜code eddr＞－〈〉 set canec vector（ ） SWAF \(4: 4000+\) IUP ROT SWAF ！ 2＋ 3400 SWAP ！；
：SU．SET．MACRO（＜SU loc＞－〈〉 set \(5 U\) for interrupt routines（ ） ASSEMBLER SU SWAF（MOU，S \(14 T \mathrm{SU}\) I）MOU，S2T SU I）RP MOU， RF 16T SU I）MOU．FORTH：
：CREATE．CAMAC．INT．WORL（＜SU loc＞＜slote＞－＜＞，creste int．（） （ defin．word．（\＃）
＜EUILDS 0 ，hERE SETUF．INT HERE 2－REG．SAUE．MACRO
SWAF SU．SET．MACRO XEQ．MACRO
DOES ［ CCOMFILE］INSTALL SWAF ！；

（ Interrupl task area initialization
r2k 16SEP81）
2030047 RLDTASK 1TASK（ Ereate task area（） 1 TASK TCLEAR（ initislize task srea＊） 1TASK DUP ！SU DUP ！（ delink task from tesk list＊） 1TASK DISPATCH（ mark tesk es ective＊）
（ ereate a ready to fire handler word for CAMAC slot 6 ） ITASK 6 CREATE．CAMAC．INT．WORD RDY．WDRD
（create fire time word for the A／D module（） 1TASK XAD CREATE．CAMAC．INT．WORD FIRE．WORD

35



RDY，WORI RAY．INT（ make RDY．INT the ready to fire＊）
（ interrupt service routine＊）
FIRE，WORD FIRE！（ make FIRE！the fire time interrupt haridler＊）

\title{
OPTIMIZED DATA STRUCTURES FOR HARDWARE CONTROL
}

\author{
Joseph D. Sawicik \\ Leboratory for Laser Energetics \\ Univaruity of Rochester
}

\section*{Abetract}

Data structures have been developed to more easily control hardware. A disk driver is used as an example for exploring alternative FORTH date structures and ways of optimizing them. These examples show that FORTH data atructures are well suited to minimizing programming time and increasing software efficiency.

\section*{Introduction}

While working at the Laboratory for Laser Energetics this summer one of my projects was to write a general purpose backup routine for a DEC-like \({ }^{1}\) RX02 mode floppy dick drive. In doing this certain commonly used FORTH toole becarme useful. This paper will serve to illuatrate these tools, and the modifications necessary due to the nature of the project.

\section*{Data Structures}

The TO concept was developed by Dr. Paul Bartholdi and was described in FORTH DIMENSIONS Vol. I No. 4 and Vol. I No. 5 concept \({ }^{2}\) in variables. This could be implemented in high level as follows:
```

O variable %to
: TO 1 %TO :
: VAL <BUILDS ( <\$>-<> , ACCEPTS INITIAL VALUE )
DÓES> (<\#>-<>;<>-<\#>, STORES OR GIVES "VAL" )
KTO @
IF !
0 %T0 !
ELSE @
THEN ;

```

It would be used like a variable. Entering \(D\) VAL <NAME>would define a variable with an initial value of zero. To chengo the value to a six one would say 6 TO<NAME>; saying<NAME>would now put a six on the stack.

This technique makes the code more readable by sliminating the use of \(O\) and [ with varibblas (and ' with constants) to acceas and modify them. The backup triver is no exception to this and in fact offers the opportunity to carry the concept one step further. In the DEC PDP-11 architecture, VO is memory mapped so that, for instance, the Disk Control Status Register is at location \(1771700^{3}\) and the Date Buffer Register is at location 1771720. One way to communicate with these addresses is to define two constents:

\section*{1771700 CONSTANT CSR \\ 1771720 CONSTANT OBR}
but then the use of Q and [ becomes necessary. A wey around this problem is to define a data structure similar to VAL except that it contains an address in its parameter field instead of a value. It would also be useful to fetch the addreas as well as to send data to and from the address. An easy, though by no means optimal, implementation of ach a structure is given below.
\[
\begin{aligned}
& \text { ( TEST BEN FOR BEGINING OF RXO2 DRIUER JOS 15JUNG: } \\
& \text { : REGISTER SBUILDS } \\
& \text { IOES, (GIUES REGISTERATA TYPE CALLED A REGISTER CONTENTS OR SENDS DAT } \\
& \text { TO THE REGISTER DEPENDING ONTHE STATUS OF YMO } \\
& \text { DUP } 2 T 0 \text { ( GET ADDRESS OF REO ANB XZO } \\
& \text { DUP }-1=\text { IF SWAF SHAP (GET CONTENT) } \\
& 1=\begin{array}{l}
\text { IF } \\
\text { THEN } \\
\text { THTORE VALUE IN REG: } \\
\text { : }
\end{array}
\end{aligned}
\]

Once these two structures are implemented it becomes very easy to talk to the diak drive. For example, if val hed been defined called \(\mathbb{N}\)-TRACK

In the RX02 mode there are eight disk commands. They are all similar in that they need to have a drive and density bit set and they are sent to the CSR. The first problem is solved by a VAL called DRIVE/DENSITY and the four words shown below:
```

: SINGLE-LIENSITY ( COM.-<COM., SETS THE :ENSITY BIT TO 0)

- IINJEEDIENSITY 25S EIC TO LIRIVEIDENSIIY O NSNSITY BIT 10, ,
: IOURLELDENSITY, CON.O-COM. SETS THE DENSITY BIT TO 1)
LKIVE/DENSITY 25% FIS TO LRIUE'DENSITI;
: ODRIVE, COM..-COH., SETSSTHE LINIVE SIT TO O)
DKIVEIIENSIIY IONIC TOLIFIVE/DENSITY:
: IDRIVE COM.,- EOM., SET YHE DKIVE EIT TO 1,
DRIVE/UENSITY io HIS TO URIUE/DENSITY:

```

After setting the drive and density as desired, the VAL DRIVE/DENSITY can then be ORed with the command to produce the desired results. There are two approaches that can be taken at this paint. For example, take the command to format a disk in a single or double density; call it (SET-DEN). A word could be defined, along with seven others like it, ss shown:

\section*{: (SET-DEN) 110 DRIVE/DENSITY OR TO CSR ;}

The second approach would be to again use a defining word:

> : IISA-COMHANI SUILIS ( \(C\) CON - - TANES THE CON FOK A [ISN OF. )
> HÓSS (GEY COH ANS TIRIVE HEN INFO UR, AND SEND)
> Q IfIUE IENSITY OK TO CSK :

110 [ISK-COMMAND (SET-DEN) ( USED TO FORHAT IISKS SING DK D UEN)

\section*{Optimization}

As usual we have a classic FOR [H space-time tradeoff. The second approach executes somewhat slower (see figure l) because the constant needs to be fetched, but whereas the first approsch takes 18 bytes per command or a total of 144 bytes, the second approach takes only 10 bytes per command plus 24 bytes for the defining word for a total of 104 bytes. Because of the space savings the philosophy that very similar things should be grouped together could override the execution speed losses and the second approach was used.

All of this would have been fine except that when doing the track to track backup a sector interleaving technique must be used to keep backup times down to a reasonable level. Since these VAL's and REG's have high level IF statements in them and they are used each time a sector is read or written, they require an overly large interleave step size. The solution to this problem is to use ;CODE instead of DOES* Though this makes the word less transportable it isn't seen as a problem since this is a PDP-11 specific disk backup. The VAL word now can be defined as follows:
\[
\begin{aligned}
& \text { ©CODE ( OK GETS VALUE OR STORES VALUE ) } \\
& \text { ZTO F TST, (SEE IF \%TO FOSITIVE) } \\
& \text { GT IF: } \\
& \text { MFARAM W I) S ) + MOV, ( STORE VALUE ) } \\
& \text { ZTO F O MOV. ( ZEKO OUT ZTO FLAG } \\
& \text { ELSE, } \\
& 5 \text {-1 WFARAM } \cup \text { I) MOV. (FETCH VALUE OF VAL) } \\
& \text { THEN: NEXT. }
\end{aligned}
\]
where \(W\) is the PDP-11 register containing the CFA (code field address) of the word executing, WPARAM is a constant equal to the offset from the CFA to the PFA, and D) indicates indexed addressing. Not only is the coded VAL faster than the high level version, but it is also faster than a VAR at fetching and the same speed at storing (see figure 2). It was also necessary to code REG as shown below:
```

; \&EG rEUILIIS ( EUILHS A IMTA TYFE CALLED A REGISTER )
\#COLUE (*)
%TOF TST, (CHECK IF XTO IS POS NEG OK ZERO)
GTIF,
WFARAM \ EI) S )+ MOV, ( STORE VALUE IN REG !
EL5E;
LT IF,
S -) WFARAM ( EI) MOV, ( GET VALUE)
ELSE, S -) UFARAK H I; MOV, i PUT T.O.S.,
THEN,
HEN,
ZTOF O \& HOU, NEXT.
ON il!

```

To illustrate the use of these concepts the FORMAT-DISK word will be shown. But first to insure that the program doesn't try to do things before the disk controller is ready, two more words are needed that wait for the done and tranafer request bit to be asserted in the こSR.
```

: TK.WAIT (WAITS FGR THE [AATA TRANSFER EIT TO BE SET)
EEGIN 2OOO FRON CSR ANDEND G

- HONE. WAIT ( WAIIS FOK THE DONE HIT TO BE ASSERTEII)
EEGIN 400 FROM CSK AMII ENII;

```

The disk command as shown before was called (SET-DEN). After recoiving this commend the dik controller waita for a "key" byte (1110, the letter I in ASCII) to be sent to the DBR, therefore the entire commend is coded as showns
```

: FOKMAT-DISA(`-@,SETS THE DENSITY OF ( DISK)
(SET-EEW) TK.WAIT
(SEI-GENDTK,WAIT 'KEY GYTE)
DONE.WAITP;

```

To format the disk in the trive one double denaity one would enter IDRIVE DOUBLE-DENSITY FORMAT-DISK; to format the disk in drive zero single denaity one would enter ODRIVE SINGLE-DENSITY FORMAT-DISK.

\section*{Tirning}

To show the effects of the different approaches timing testa were run. The first contraste the difference between the two types of disk commands. In all tests the action was placed inaide a double loop likes
: TEST 10000300000 DO LOOP LOOP ;
This routine took 23 seconds which was then subtrected from the other results to give the time to do the operation 300,000 times. This was then divided by 300,000 to give the time per operation. Theas are the remulte on a DEC LSI 11/2:

To Sead Disk Comand
\begin{tabular}{ll} 
Colon definition & .23 msec. \\
Defining word & .28 nsec.
\end{tabular}

Then a high level VAL was compared to e coded VAL and a VAR:
\begin{tabular}{lcc} 
& fetching (esec) & storing (esec) \\
high level VAL & .237 & .39 \\
coded VAL & .067 & .11 \\
VAR & .083 & .093
\end{tabular}

\section*{Surnmery}

This paper not only showed the usefulness of certain techniques in FORTH but also illustratea sorne general properties of the language. The first of these is the ease of implementation of new date atructures. Through the use of BULLDS ... DOES or BULLDS ... ;CODE one cen first build the structure to suit the needs of the epplication end then imbed in the executable code neceseary operations for the structure. Also atructure cen easily be given varieble execution an in the case of VAL. end REG. Another impartent benefit of FORTH is the ease of optimization of the word by the use of asembly code. Chenging the VAL end REG words to ;CODE took lese than a half hour.

\section*{Acknowiledemmenta}

I would like to thank Mike McCourt, Bob Keck. Lewrence Foraloy end Petar Pauleon for their help in getting the hardwere rurning end for commente on the software.

This work was partially apported by the following sponsors: Exxon Research end Engineering Compeny, Ceneral Electric Company, Now York State Energy Research and Development Authority, Northeact Utilitioe, The Stendard Oil Compeny (Onio), the University of Rocheater, Empire State Electric Energy Research Corporation, end the U.S. Department of Energy inertial fusion program under contract number DE-AC08-80DP40124.
J. Sawicki is an undergraduete with the Electrical Engineering Departrnant of the College of Engineering at the Univeraity of Rochester. He is a DJ in his spare time.
1. DEC and POP-11 are tredemarks
2. The TO concept by Paul Bartholdi FORTH DIMENSIONS Vol. I No. 4 and Vol. I No. 5.
3. Where an O miffix indicates octal

\section*{THE STRING STACK}

\author{
Michael McCourt \\ Laboratory for Laser Energetics \\ University of Rochester \\ Richard A. Marisa \\ Production Autnmation Project \\ University of Rochester
}

\section*{Abstract}

Applications which require a text data type are supported by a group of functions which operate with string variables and a string stack. The string stack is analogous to the parameter stack, however, the data type with which it operates is the string, containing length and character data.

\section*{String Defining Words}

Two defining words are available for the creation of string data entities. The first is:

> <maxlen> STRING-VAR <NAME>
which creates a varying length character string with maximum length <maxlen>. Invoking < NAMExplaces
<beginning address><aaximua string length> on the parameter stack. The first byte at <beginning address>is the current string length; the string text begins at the next byte.

The second string defining word is: <number of elements> <maxlen> ()STRING <NAME>
which creates an array of variable length strings. Invoking
<i><NAME>
places <address of the i-th string> <maxien>
on the parameter stack. Note that (number of elements) \(x\) (maxlen) bytes will be allocated to hold the string array.

\section*{String Stack Manipulation}

A string stack, separate from the parameter stack, is maintained in memory for the purpose of manipulating string data. Several words which manipulate the string stack are defined in the string stack library which cen be compiled by executing >STRINGS (which loads in the string stack package). Currently 200 (decimal) bytes are allocated for the string stack.

The quote word (") is available for placing a string on the string stack. To stack a string, type:

> "<text>"
" is followed by exactly one space, then <text〉 delimited by a quotation mark.

A string print word . \(5 S\) is used to print the top element of the string stack,

\section*{}
© STRING STACR--FIXED LENGTH STRING COMPARISON LAR 19-SEP-79

 COMFARES CHARS. IN STKINGS A A PARIUISE: KETURNS O IF ( COAFARES CHARS. IN SIRINGS A AF AFA
: STFIO O SHAP O IIO IRROF OUER CE OUER CE - ROT It ROT \(1+\) ROT DUP \(0=\) NOT IF LEAUE THEN LOOP;
[ADD A, ADU B, LEN]-- \([=O R+O R-]\), SAME AS S?F[1O)
(EXCEPT ADDRESSES NOT RETUKNEA)
: S?F S?FLO ROT ROT 2UROF:
([ADDAHLEN]——E=OR \(\ddagger\) OR - ], COMFARESSTR]NG ATO) : STB O SURING OF BLANKS - RETURNS O IF THO AKE EQUAL SOF IUP CE BL SWAF I SNAF DUF O

IF LEAVE THEN LOOF SHAF DROF:

\section*{}
( STRING COMPARISON---VAFYING LENGTHS )
([ADI A, ADI I, LEN IIFFJ--I = OR \& OR -] FIRST TESTS)
(TO SEE IF LENGTH IIFF. GTHN. A E IS O I IF NOT. TESTS)
( THE LONGER STRING TO SEE IF THE EXTKA CHAFS. AKE HLANKS
(IN BOTH CASES O IS RETURNEI, OTHERWISE + OF - )
: STGLTEST IUUF O = IF DROF 2LIROF O ELSE IUF O
IF HINUS FOT DROF STH MINUS ELSE SHAF DORP S?E THEN THEN
(AIII A, ALID BJ---[OIF A=E, IF A, B, + IF A;E J TESTS)
WHETHEF Z UAFIABLE LENGTH STRIGNS NAUE DOTH THE SAME )
OF CHARS. ANI THE SAME ORDEF \& TYF'E)
: \(5^{7}\) OUER CE OUER CP 2OUF - WR MINROT It ROT \(1+\) ROT S?FDO DUF OK IF KOT ROT 2DROF RY IIROF

98

STRING STACK HORIS
SUAK SSO O SUAR SSM O SUAR SST
SSTOF SST \(Q: ~ S S A O F!~ S S T ~: ~\)
([FROH, TO, LEN J---['] CHECAS FOR STACK BOUNDARIES)
SDUCHECK OUEF SSORG U.
IF SSHAX SSTOF! 14 T TABORT THEN
([ADI]--[] INSURES THAT ADDKESS POINTS IO STKING)
SSUER IUE DUP CE + SSMAX US=
IF SSHAX SSTOP \(13 T\) TABORT THEN;
(SADI OF TOF STKING]--[AII OF NEXT STRING DOWN])
SSUOWN IUUF CE It
SSFUSH DUF CE \(1+\) SSTOF OUER - DUF SSTOF! SUAF RMOUE ;
-

STRING STACK WORIS LAR 19-SEP-79)
- DROF SEUER SSIOL] SS[] REMOUES TOP STRING FROH STACK ( )

SSTOF SSUER SSDOWN SSTOF:
[LEN ( [J]--[] RETURN LEN OF TOS STRING *)
SSTOF SSUER CE ;
: LOC \(1+\); ( \(]---[]\) RETURN ALIIR OF TOS STRING *)
- LUU ( []--[] COPY TOS STRING ()

SSTOP SSUER SSFUSH :
-SUAF DUF SSDOUN [JUF SSPUSCHANGE TOF 2 STRINGS SOUN SSTOF SUAF SSTOF! SWAF 5SPUSH SSPUSH ;
 SSTOP UUP SSDOHN DUF SSDOUN DUF SSFUSH SSDOWN SSTOF SHAF SSTOF! SWAP SSPUSH SWAF SSFUSH SSFUSH;


( STRING STACK WORLIS MAH 13-JUN-80 )
: DUER ( []---[] PUSH 2ND STRING DONN ONTO TOS ()
SSTOF SSDUHN SSUER SSPUSH:
: 2DUF
- aver cover
- grof jrof

SSTOF SSDOUN [ ]---[] PUSH 3RD ANU ATH TO TOS \%)
SSIOF SSDOUN, SSGOWN [JUF EXCHANGE IST \& \(2 N D\) UITH 3 KD AND 4 TH ( ) DUF SSIOUN SSDOUN SSDOUN SSTOF! SSPUSH SSPUSH SSPUSH SSPUSH :

removing the top element in the process. For example,

OK " STACK THIS STRING" <CR>
OK
.SS <CR>

\section*{STACK THIS STRING OK}

Notice that the functions . 55 and . are similar. Several other functions operate on the string stack in a mamer analogous to words which operate on the parameter stack. These are:

\section*{}
( STRIMG STACK MORDS CONT'D MAN 13-JUN-60)
: '! ( [ADR][LEN]---[] STORE TOS AT ADDR. 1 DKOF TOS () SSTOP 'PROP SUAP OUER CE MIN 2DUF SUAP C! It
OT SUAF RMOVE:
[STRINGJ-K-[J STORES STKING IN PAD THEN NOUES IT FROM)
( THERE TO THE TOSS -- MORKS DURING EXECUTION TIHE)
x: 420 MORD o \({ }^{\circ} \mathrm{R}\) \&
\$' R R DUP O 'e DUP CE DUP 2 MOD
( LSTRING]-[J STORES STKING AT TOP DF DICT. STACK )
( DURING COMFILATION)
C.COHPILE \(1+120\) NOKD Ce DUP 2 NOD
- IF STATEESE 2+ THEN ALLOT :
: - State if C' ELSE X' Then ; imp.
\begin{tabular}{|c|c|c|c|}
\hline WORD & FUNCTION & BEFORE & AFTER \\
\hline "DUP & copies top of stack & B A & B A A \\
\hline "SWAP & reverses top two & B A & \(A 8\) \\
\hline "DROP & removes top of stack & B A & B \\
\hline "OVER & copes 2nd string onto top & B A & B A 8 \\
\hline 'ROT & moves 3rd etring to top & C BA & B A C \\
\hline "2DUP & copies top 2 strings & B A & B A B A \\
\hline "2DROP & removes top 2 strings & C B A & C \\
\hline "2SWAP & reverses 1\&2 with \(3 \& 4\) & DCBA & \(B A D C\) \\
\hline "20VER & copies \(3 \& 4\) to top & DCBA & DCBADC \\
\hline "+ & string addition (catenation) & B A & BA \\
\hline
\end{tabular}

\section*{String Reletionals}

Just as the parameter stack relational operators remove their argumente from the parameter stack, the following atring stack relational operatore remove their arguments from the string stack. The logical result of the string relation is placed on the parameter stack. The availsble relationals are:
\[
\begin{array}{ll}
"= & "<= \\
"<> & ">= \\
"< & ">
\end{array}
\]

String Variable Stornge and Retrieval
The string atore word, "L, places the top of the string atack in the string variable described by the parameter atack, popping the string stack. The string retrieve word, "@, places the string referred to by the parameter stack onto the string stack.

OK 30 STRING-VAR MYSTRING <CR>
OK
"string text " MYSTRING "1 <CR>
OK
MYSTRING "@ HYSTRING "@"+.sS <CR> string text string text
OK
" string text " MYSTRING "1 <CR>
OK


( STRING STACK WORIS CONT'D MAM 10-MAR-81)

: SST ( \(>-\langle>\), PUF STRING IN DICTIOMARY, MAKE EUEN LENGTH; 420 WORD COUNT DUP HERE SUAP If -2 AND ALLOT SWAP CHDVE;
( SOME FIXED LENGTH STRING DEFINITIONS )
( [ADDRFHEX LENJ-E-[] PUSH STRING AT ADORR TO TOSS)
: © RF DUP SSTOP OUER - 1- SSTOP! SSTOP CI SSTOF
\(1+\) SWAP CMOUE:
: © if 2 DDUR MAX LENJ---[] COPY CHARS OMLY FROM TOSS TO ADDR )
-->

( STRING STACK YORDS CONT'D LAR 19-SEP-79)
 SSTOP C! SSTOP DUP \(1+\) ROT It RHOUE SSTOP it SSTOP! :
( CLEN, BEGINNING CHAR \%)---[J REPLACE TOSS HITH) (SUBSTRING OF LENGIH GLEE-[J REPLACE TOSS HITH)
CHAR OF ORIGINAL STRING). STARTOHG WITH SPECIFIED ) : 'SUBSTR I- SSTOP SSUER SEROF + DUP ROT ROT C! SSPUSH: ( LADD OF 2MD STRT1, \(15 T\) CHAR OF \(1 S T\) STR, LEN DF \(2 N H\), O J)
 : INDEXDO DO OUER I + CO OUER =
IF OUER I \(48 S T O P ~ 1+~ L E N ~ S P F ~\)

- －aking the name of the string variable －STRING in the preceding example ：ased＜address＞＜moxien＞on the para－ －eier stack．String store and string －：：eve check the maximum and current a－g！h of the string variable when moving \(\cdot \rightarrow\) g data．

Nhen it is required to move fields of ：Ef length which do not contain an ＊－＝edded current length in the first byte， sed length string store and retrieved －：－Is may be used．The syntax is：
\[
\begin{aligned}
& \text { <address> <length> "!F } \\
& \text { <address> <lengrh> "@F }
\end{aligned}
\]
ixing Functione
－EN returns on the parameter stack， \(\because\) ength of the string on top of the － 3 stack．The string remains on the －-7 stack．The address of the firat byte ：he string（one byte after the length z＝is found by executing＂LOC．
e－gth＞＜beginning character number＞ ＂SUBSTR
a．ases the top of the string stack with a ＊．：s：－ing of length＜length＞，beginning －：－the specified character of the －nal string．For example，

OK
＂abcde＂ 23 ＂substr ．SS
cd OK
The＂INDEX function searches for the
\(\because\) ：occurrence of the string in the e：and string．If an accurrence is found，
affset is returned on the parameter －ミこ．．If an occurrence is not found，-1 is ？ined．The top of the string stack is ：：reed．
inng Stack Errors
－wo errors are reported by the string ¿＝к package：string stack underflow and ：－flow．As stated previously 200 bytes \(:=\) initially allocated for the string ：\(-\alpha\) ．If repeated overflows are gener－ \(: \div\) more space can be allocated for the －\(Э\) stack by changing the parameter i＝ssed to＂INIT in the string atack ：－ary．String stack initialization is the z：function performed when the string －：ex library is loaded．
inmmary
This was the first major software \(\because=\) age transported throughout the －versity URTH community．Originally， －ad a few code routines which were －̇こhine specific to reduce execution －e．However，these were removed on z ：he systems but the Intel 8080．The ：\＃こxage has run，without change（except \(\because\) the above mentioned machine－specific ：こse＇on Hewlett Packard 2100，DEC \(=こ 2-11\), IBM 360 and the INTEL 8080.


TRING STACK HORIS CONT＇I
（［］－－－［－1 OR OFFSET］SEARCHES FOH IST OCAR 19－5EF－79 TOP STR．IN 2ND STK，－－－IF FOUND OFFSETIS RETURNEI ）
－INDEX－ISSTOP DUP CE ORE ROT \(1+\mathrm{CE}\) ROT \(1+\) SWAF
ROTO O INDEXDO
ELSE O ROT ROT THEN 2OROP DROP ；
－SJTOF［J COMFARE SNROP TOP 2 STRINGS，LEAUE \(0 .<0\) OR \(\geqslant 0\) ）
（［J－－［T／FT LOGICAL \(:\) ，IESTS TOF 2 STRINGS）
（\｛？\(\left\{\begin{array}{l}0 \\ 0\end{array}\right.\)

STRING STACK LORDS CONT＇D MAN 18－MAK－81
－（［J］－－［T／FJ TESTS TOP 2 STRINGS FOR ；＊）
－\(\%\)
：\(\%\) NOT
\(\because\) NOT
－SPACES
 SSTOP I－DUP ROT SMAF C！SSTOP！；
 17 HERE

STKING VARIABLE AND STRING AKRAY （［HAX LEN］－－－［］ALLOTS SFACE IN DICT FOR MAX LEN ANG） S MAX OF CNARS：
（［MAX LEN］STRING＜NAME）－－－BUILIS A STRING VARIABLE WHEM SNAME IS EXECUTED THE BYTE ADIRR．OF THE STAING） START AND LENGTH AKE LEFI ON THE STACK
STODE 5 －\()\) SULIIS STKING－SPACE

（STRING ARRAY ROUTINE MAM 13－JUN－80）

O DO DUF STRING－SPACE（ALLOT DIC SFACE，STORE MAXLEM） LuOP DRUP

BUE 2 MOD ROT ROI 3 PICK TMEM AgDR OF 15 EL ENENT ROT \＆ \(2+\)＋SHAF ；（STRIMG ADDR IF EUEN；2t FOR MAXLEN）
（RETURMS COUNT ELENENT OFFSET）
\(-->\)
（ STRING EXECUTION ROUTINE LPF，MAM 18－MAR－81）
：EXEC（＜HORD MAME ON TOSS〉－く〉，EXECUTE HORD IF FOUND NERE LEN •！ ELSE O TABORT THEN ；（ UNDEFINED YORD ERROR ）
－FORGET（＜UORD NAME ON TOSS〉－く〉，FORGET NORD IF FOUNE ） HERE LLEN I！UPARAM＋SFORGET ELSE O TABORT THEN ；（ UNDEFIMED WORD ERROK ）

The first application was for a screenoriented data entry system, Later applications included an ISAM data base, a menu-driven interface for flow cytometry and a ward processing system. The package consists almost ontirely of its original code written in 1977 by Mike Williams, of the University Computing Center. The major change has been the addition of comments.

\section*{Acknowledgements}

We would like to thank the following people for their assistance: Mike Williams, of the University Computing Center, who developed the original String Stack Package for URTH on the IBM 360 and the intel 80B0; and two undergraduates who worked for Lawrence Forsley, Lynn Raymond and Den Blumenthal, for documenting this package.

This work was partially supported by the following sponsors: Exxon Research and Engineering Company, General Electric Company, New York State Energy Research and Development Authority, Northeast Utilities, The Standard Oil Company (Ohio), the University of Rochester, Empire State Electric Energy Research Corporation, and the U.S. Department of Energy inertial fusion program under contract number DE-ACOBBODP40124.
R. Marisa is the manager of the computing facility of the Production Automation Project in the College of Engineering at the University of Rochester. M. McCourt was a senior laboratory engineer with the Laboratory for Laser Energetics at the University of Rochester and is now an applications engineer for Harvey Electronics.

\section*{HELP WANTED \\ Amociate Syuterna Maneger, Pulmonary Computer Syaterne}

Primary responsibility for designing, debugging and implementing major software projects on the Pulmonary Computer System. Programming experience with PDP-11 Assembly language and FORTH desirable. Some hardware experience will be useful.

Salary range to \(\$ 35,000\). Superior benefits package, three weeks vacation first year.
Contact:
John Gilbert, Employment Officer
Cedars-Sinai Medical Center
8723 Alden Drive
P.O. Box 48750

Los Angeles, CA 90048
(213) 855-5529

\section*{NEW PRODUCTS}

\section*{FORTH Application Moculos Dinkette}

The diskette of FORTH application moduels, 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 deta structures, software development aids, atring manipulators, an expanded 32-bit vocabulary, a screen calculator, a typing practice program, and a manu generstion/selection program. In addition, the diskette provides examples of recursion, <BUILDS...DOES> usage, output number formatting, assembler definitons, and conversational programs. One hundred screens of software and one hundred screens of instructional documentation are supplied on the diakette. 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 \(\$ 75\) (if other than 8 " standard diak, add \$15). To order the FORTH modules, write Timin Engineering Company, 9575 Genesee Ave., Suite E-2, San Diego, CA 92121, or call (714) 455-9008.

INANER ACCESS FORTH SOF TWARE ANO DOCUMENTATION

Fig-FORTH compiler/interpreter for PDP-11 for RT-11, RSX11M or standalone with source code in native assembler. Included in this package are an assembler and editor written in FORTH and installation documentation.

This is available on a one \(8^{n}\) single density diskette only. \#20011-01 (\$80)

Reference Manual for PDP-11 figFORTH above. \#20011-99 (\$20)

Fig-FORTH compiler/interpreter for \(C P / M\) or CROMEMCO CDOS 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.

All diskettes are single density, witr 5.25" diskettes in 128 byte, 18 sector/track format and \(8^{\prime \prime}\) diskettes in 128 byte, 26 sector/track (IBM) format.

Released on two 5.25" diskettes with source in 8080 assembler \(\%\) 20080-85 (\$80).

Released on one \(8^{\prime \prime}\) diskette with cource in 8080 assembler \({ }^{\text {\# }}\) 20080-88 ( \(\$ 80\) ).

Released on two 5.25" diskettes with source in Z 80 assombler (20080-Z5 (\$80).

Released on one \(8^{\prime \prime}\) diskette with source in \(\mathbf{Z 8 0}\) assembler \#00080-28 ( \(\$ 80\) ).

Manual for CP/M (or CROMEMCO) fig-FORTH above \#20080-99 (\$20).

METAFORTH Cross-Compiler for CP/M or CROMEMCO CDOS to produce fig-FORTH on a target machine. The target can include an application without dictionary heads and link worda. It is available on single density didettes with 128 byte 26 sector/track format. Target compiles may be readily produced for any of the following machines.

CROMEMCO--all models
TR 580 Model II under CP/M
Northatar Horizon
Prolog \(\mathbf{Z 8 0}\)
Released on two 5.25" diskettes 20100-85 ( \(\$ 1,000\) ).

Released on one \(\mathbf{6}^{\text {n }}\) diskette \(\mathbf{2 0 1 0 0 - 8 8}\) ( \(\$ 1,000\) ).

Complete Zilog (AMD) 28002 develoment system that can be run under CP/M or CROMEMCO CDOS. System includes a METAFORTH Cross-Compiler which produces a 28002 fig-FORTH compiler/interpreter for the Zilog 28000 Development Module. Package includes a 28002 assembler, a Tektronix download program and a number of utilities.

Released on two \(5.25^{\prime \prime}\) dickettes 29102-85 ( \(\$ 4,000\) ).

Released on one \(8^{\prime \prime}\) diskette 29102-8B \((\$ 4,000)\).

Zilog Z8002 Develoment Module figFORTH ROM set. Contains fig-FORTH with 28002 assembler and editor in 4 (2716) PROMS. 38002-00 (\$850).

For orders and further information, contect:

\section*{INNER ACCESS CORPORATION \\ Software Division \\ Box 888}

Belmont, CA 94002
(415) 591-8295

\section*{ANNDUNCEMENTS}

Sym-FORTH Newsletter now available, contact: Saturn Software Ltd., PO Box 397, New Westminister, British Columbia, V3L \(4 Y 7\), CANADA.

\title{
COMPLEX ANAL YSIS IN FORTH
}

\author{
Alfred Clark, Jr. \\ Department of \\ Mechanical Engineering \\ University of Rochester
}

Juring my years as an engineering educator and a researcher in theoretical fluid mechanics, I have often wished for the perfect calculator--a compact machine which would perform intricate and useful mathematical tasks in response to a few keystrokes. The pocket scientific calculators, amazing as they are, never seemed to have quite the power and flexibility (and certainly not the graphics ability) that I hoped for. I always supposed that my hopes were unreasonable until I discovered FORTH two years ago. Having been a FORTRAN programmer for 20 years, 1 found the transition to FORTH somewhat difficult and even painful at times. Originally, 1 took up FORTH out of curiosity, but gradually I realized that the quest for the perfect calculator was over--it is FORTH plus a microcomputer.

Perhaps I should say a little more about what a perfect calculator is supposed to do. Arnong other features, it should have (1) standard trigonometric and exponential functions, (2) other common special functions (e.g., Bessel functions), (3) graphics and automated plotting of functions, (4) numerical integration, (5) a root-finder, (6) special purpose applications, such as a direction field plotter for first order differential equations, and (7) complex arithmetic, including complex transcendental functions. Further, all procedures should be executable with a few keystrokes.

The last item in the list--complex--is in some ways the most stringent test of any would-be perfect calculator. It's certainly not available on any pocket calculator. Although it can be implemented in BASIC, it is cumbersome and requires a large package of subroutines. The versions of FORTRAN available for small machines generally omit the complex arithmetic and complex functions which are available on large machines. With FORTH, however, the extension to complex from real floating point is simple to implement, easy to use, and powerful. Since complex arithmetic is not yet very common in FORTH on small machines, I thought it would be worthwhile to sketch briefly my implementation.

The most fundamental question in introducing complex analysis is how to represent complex numbers. Here it turns out that the pure mathematician's definition of a complex number as an ordered pair of real numbers is exactly what we need. Thus the complex number \(3.5+7.2 i\) is regarded as an ordered pair, and is pushed on the stack by typing 3.57 .2 . With this convention established, it is easy
to define all of the important stack manipulations such as ZOROP, ZDUP, ZOVER, ZROT, and ZSWAP, which perform exactly like their integer and floating point counterparts. The basic load and atore operators, 2 @ and \(Z[\), can be defined in terms of and [.

There are many single number operations which are useful. These include the real part REZ, the imaginary part IMZ, the complex conjugate CONJ, the modulus \(/ Z /\), the square of the modulus \(/ Z / 2\), the reciprocal \(1 / Z\), and the phase \(A R G Z\) (radians). Most of these are quite aimple to define. IMZ, for example, is just
: IMZ FSWAP FDROP ; where FSWAP and FDROP are floating point stack operations. As another example, consider \(1 / 2\) defined by : \(1 / 2\) zous /2/2 FROT FONER F/ frot fror \(F /\) CON ;
For ARGZ it is very important to eatablish a precise range and to implement it carefully. The conventional range, which I have used, is -PI < ARGZ <= PI. Any carelessness in the definition of ARGZ will lend to disasters later when multi-valued functions are introduced. Many engineering applications require the phase in degrees, and it is convenient to build in a function DARGZ which supplies this.

Conversion words between rectengular and polar forms are also very useful. To go from retangular to polar, with the phase (in radians) on top of the stack and the modulus just below, we have

\section*{: POLAR ZUUP /L/ FROT FROT ARGZ ;}

A similar word, DPOLAR, leaves the argument in degrees. For corversion from polar to rectangular, we have RECT (angle in radians)
RECT FOVER FOVER COS F* FROT FROT SIN F* ;
and a word DRECT for the angle in degrees. A very useful application of these is a rotation operator ROTZ, defined so that the sequence \(Z F\) ROTZ rotates \(Z\) by \(F\) radians and leaves the result on the stack. The definition is
: ROTZ FROT FROT POLAR FHNT F+ RENT ; .
There are several different useful formats for complex output. (" "y system has 8 different formats, which is handy but a little extreme.) The word 2. prints the number as an ordered pair -- 3.57 .2 , for example. The conventional mathematical notations is obtained by Z1. -- (3.5) + (7.2)l. Words to print in polar form are also useful. For example, \(Z P\). is defined so that the sequence 3.5 7.2 ZP . gives
\(M O D=8.00562303 \quad\) ARG \(=1.11832144\) (RAD). All of these output words are defined in terms of the basic floating point print word F. . For example, Z. is defined by

\section*{: 2. FSTAP F. 2 SPACES F. :}

The binary complex operations are \(Z_{+}\), \(Z-, Z^{*}\), and \(Z /\). These are quite easy to
define. For example, \(Z+\) is defined by
: \(\mathrm{Z}+\) FROT P4 FROT FROT F + FSNAP :
where FROT is a floating point ROT, and \(F_{+}\)is a floating point add.

Higher functions can be defined, provided the underlying real floating point has the standard real functions SIN, COS, ATN, and EXP. The complex exponential, for example, is then defined by
: ZEXP PSWNP EXP FLUP FRON FOUP OS FROT F*

\section*{FROT F* FPON FROT SIN F*}

Other useful functions wuch as ZSIN, ZCOS, ZTAN, ZSINH, ZCOSH, and ZTANH are defined similarly.

Of the multi-valued functions, the most useful are the square root ZSQR, the logarithm ZLOG, and the power \(2 * *\). As an example of the definitions, consider the principal value of the square root:
: ZSOR POUAR 2. F/ ESNRP SOR FSWMP RESOR :
The basic words described sbove can be the building blocks for substantial applications. One such application, which is particularly useful pedagogically, is conformal mapping. I have defined a word MAP such that the sequence
```

MAP <curve> <function>

```
will take any previously defined curve in the Z-plane and eny previously defined complex function, and produce a graph showing the curve and its image under the transformation. This tool allows students (and the instructorD to improve their understanding of the geometry of complex functions.

\section*{Notes on Implementation}

The code described sbove runs on the author's 48 K Apple II. The underlying integer FORTH is the excellent version written by William Graves and distributed by SOFTAPE. The real floating point arithmetic and functions have been implemented by interfacing the SOFTAPE FORTH with the Applesoft ROM routines. The same data stack is used for integers ( 2 bytes), reals ( 6 bytes), and complex numbers ' 12 bytes). The code for the complex routines was written entirely in FORTH, and, in compiled form, occupies about 2 K . The conformal mapping code compiles to about \(I K\) additional.

ORDER NOW!
Proceedings of the 1981 Rochester FORTH Standards Conference \(\$ 25.00\) US, \(\$ 35.00\) Foreign. Send check or MO to FIG in US funds on US bank.
"Starting FORTH"
Hard - \(\$ 20.00\) US, \(\$ 25.00\) Foreign Soft - \(\$ 16.00\) US, \(\$ 20.00\) Poreign ORDER NOW!

\title{
A FORTH BASED MICRO-SIZED MICRO ASSEMBLER
}

\author{
Gregory E. Cholmondeley
}

Laboratory for Laser Energetics University of Rochester

\section*{Abstract}

The FORTH programming language can be used to implement a very amall and useful micro assembler. Functions ranging from automatic field alignment to user definable macros can be written and altered easily, permitting a flexible and easy to use micracoding technique. This paper also serves to illustrate several of the many programming features found in FORTH.

\section*{Introduction}

Computer central processors often contain an iternal data form called "microcode." This code defines the instruction set of the processor. The creation of this internal cade is called "microcoding."

Microcoding by hand is at best a tedious and wasteful undertaking where a significiant portion of a programmer's time is spent aligning fields, formatting output and correcting typographical errors. Understanding (let alone debugging) a microcode program is difficult due to the lack of readability from a human point of view. Through the use of comments, automatic field positioning, labels and other such tools, a good micro assembler should minimize the above problems making microcoding a much more agreeable form of programming.

There already are micro assemblers written which handle these along with other problems associated with microcoding, but most of them share one rather serious trawback: they are large programs. The micro assembler presented here is based heavily upon the Signetics micro assembler but requires only a few "blocks" of FORTH code. Thus it is possible to have a micro assembler on a small home computer[ Such an assembler could be used as a design tool as well as an inexpensive and effective teaching aid. It would allow even wide instruction words to be built in a simple to use, high level form.

\section*{Unege}

There are two main phases associated with this micro assembler: instruction definition and actual programming. A third phase will be implemented shortly to allow the user to explicitly and easily define output formats. The first of these phases to be explored is the instruction definition phase. This is the time when the various instruction word formats are
defined. A simple example of such a definition would be as follows:

INSTRUCTION WIDTH 8
Define an 8-bit instruction.
FIELD A WIDTH 4 DEFAULT 3 Define field \(A\) as the 4 most significant bit positions in the instruction, having a default value of 3 .

\section*{FIELD B WIDTH 2}

Define field \(\mathbf{B}\) as the next 2 bit positions, having a default value of 0 .

FIELD C WIDTH 2 DEF AULT 1
Define field \(C\) as the 2 least significant bits, having a default value of 1 .

\section*{END.INSTRUCTION}

Close the instruction definition.
The resulting instruction word would appear in the following form:


From this point on the field names \(A, B\), and \(C\) will be unique end may not be used to define other fields.

While the preceding example is rather trivial an inatruction definition may become quite complex. It is, for instance, possible to define multiple formate for every field, with each of these containing multiple sub-fields. This is useful when it is deemed that fields should have different meanings depending upon the context of the rest of the instruction word (vertical versus horizontal programming). Subfields are treated in the same manner as fields so that they too may have multiple formats and sub-fields. This feature is implemented as a tree structure ellowing an unlimited nesting of fields, formats and sub-fields. Figures (1) and (2) thould clarify this concept.

This part of the micro assembler has error checking capabilitiss which prevent unintentional overwriting of fields. For example, if field EE of figure (1) is filled, then fields BB, DD and GG (and of course EE) could not be used. Automatic field defaulting uses the same mechanism so that if field EE is the only field filled (using the format from the previous example) then fields \(A A, C C, F F\) and \(H\) will be defaulted.
\begin{tabular}{lll} 
INSTRUCTION WIDTH 32 & \\
FIELD AA WIDTH 8 & DEFAULT 255 \\
FIELD BB WIDTH 16 & DEFAULT 65535 \\
FORMAT & & \\
FIELD CC WIDTH 4 & DEFAULT 15 \\
FIELD DD WIDTH 12 & DEFAULT 4095 \\
FORMAT & & \\
FIELD EE WIDTH 10 & DEFAULT 1023 \\
FIELD FF WIDTH 2 & DEFAULT 3 \\
FORMAT.END &
\end{tabular}

INSTRUCTION WIDTH 32
FIELD AA WIDTH 8 FORMAT

FIELD CC WIDTH 4 FORMAT

FIELD EE WIDTH 10 DEFAULT 1023 FORMAT.END
FORMAT.END

\section*{FORMAT}

FIELD GG WIDTH 16 DEFAULT 65535
FORMAT.END
FIELD HH WIDTH 8
DEFAULT 255
END. INSTRUCTION
Figure (1) : Sample Instructon Definition
(nstruction

Figure (2) : Structure of Figure (1)

The programming phase of the micro aseembler is where the actual microcoding takes place. An inatruction is created by typing the name of a field followed by a number or expresaion representing the value that that field should take. This is continued for as many fields as needed in the instruction word. When the instruction is complete a "\$" (dollar sign) is typed and the computer readies itself for another word. At this point any undefined fields are set to their default values, the inetruction and other related information is stored in memory, and the location counter is incremented. Figures (3) and (4) demonstrate a simple microcoded program which merely eets one field at a time equal to a zero.

PROGRAM IEXAMPLE WIDTH 32
ORG 512
A 0 \$
BB 0 \$
CC 0 \$
DD 0 \$
EE 0 \$
FF \(0 \$\)
GG 0 \$
HH 0
END. PROGRAM

Figure (3) : Sample Program
\(: こ 000000111111111111111111111111\) \(\therefore 111111000000001111111111111111\) \(\therefore 111111000011111111111111111111\) \(\therefore 111111111100000000000011111111\) \(\because: 11111111100000000001111111111\) \(\because: 11111111111111111110011111111\) \(\because 1111110000000000000000011111111\) \(\because: 1111111111111111111111100000000\)
\begin{tabular}{|c|c|}
\hline M used & \(B B^{\prime} \mathrm{HH}\) d \\
\hline BB used & AA \& HH defaulted \\
\hline CC used & AA, DD \& HH defaulted \\
\hline DD used & A , CC \& HH defaultad \\
\hline EE used & M, CC, FF \& HiH defaulted \\
\hline FF uned & AA, CC, EE \& KH defaulted \\
\hline GG used & A \& HM defaulted \\
\hline HH used & M \& BB defaulted \\
\hline
\end{tabular}

Figure (4) : Sample Output

While automatic field alignment is in iself a vast improvement over hand soding, there are a few other tools availasle to the programmer which make microcoding even easier. A "(." denotes a : omment allowing anything up to end reluding a ".)" to be ignored. Typing ORG 3 ad a number or an expression will aet the - xation counter ( LC ) to that value. -rping SET <new variable name> TO <number or expression>
will declare and initialize a variable, while :ving EQU <old variable name> WITH <number or expression> -ill store a new value into a previously seclared variable. These variables raturn : heir value when they are typed (similar to * constant in FORTH) and can be used in expressions at any time and in any phase :" the micro assembler.

One of the most veratile tools availsole in this micro assembler is the \(\therefore\) :SROP function. Microps are userefinable functions designed to eliminate \(\equiv\) large part of the repetitious program--:ng associated with microcoding. For example there may be times when several 'eids will always take on constant or -elative values. Rather than cluttering :-e program by having to set all of these 'eids every time, a microp can be written : 2 do this automatically. A program writ:en using well named microps can in turn ze quite a bit easier to read and unders:and than one which merely sets the 'ields.

The definition of a microp requires a -nique name and a set of commands which will be executed whenever its name is zalled. Any FORTH programmer will soon ealize that a microp definition is nothing : : her than a colon definition, thus allowT 9 the full power of FORTH to be easily accessed directly from the micro assem\(=\) =er[ An example of a simple microp that sets a few fields to zero would be:
\begin{tabular}{ll} 
MICROP EX1 & (. set fields CC, FF, \\
CC 0 & and HH to 0.\()\) \\
FF 0 & \\
HH O & \\
END.MICROP &
\end{tabular}

An example of thit microp in use would be found in the programming phase and might look like:
```

MA HH (LC ) \$
AA B EXI \$

```

NOTE: LC in the preceding example is a variable, the "(" and ")" are required for ite proper execution. They do not denote comment in the MICRO vocabulary context. This is also true when building micrope. In the MICRO vocabulary comments are delimited by "(." and ".)".

Being simple colon definitions, microps can do internal testing, looping and everything else offered in FORTH. Microps can expect parameters on the atack as well as numbers or expressions from the input buffer via a function called GET*. For example:

Another way to increase readability in the micro assmbler is through the use of lebels. This foature is only partially implemented at this time but will work as follows. Labels must have unique names and must be declared via LABEL statements before they are used. When a labe! is found immediately preceding a now instruction word (or in other words; immediataly following a " \(\$\) ") the curren: value of the location counter ( LC ) is stored as the value of the label. Muitiple Iabels may be used to represent the same line of code. Whan a label is used inside an instruction definition after its velue has been set, it will be treated as any other variable. If the lebel has not been set to a value (i.e., forward referencing) a \(z e r o\) will be returned and all information neceseary to resolve the reference will be stored in memory for the second pass. During the second pass the micro assemblar will shift the correct value(s) of the label( s ) into the proper place \((\mathrm{s})\) and then edd the resulting number to the rest of the word. This allows labels to be referenced more than once in a single inatruction. It also allowa addition and subtrection of other non-label expresaions to labels (i.e., AA ( \(1 \mathrm{LABEL}+2\) ) or AA ( ILABEL - 1) but not AA (1024-1LABEL )). When this is implemented another extended precision function ( \(E_{+}\)) will be needed to perform the extended precision addition.
\begin{tabular}{|c|c|}
\hline \multicolumn{2}{|l|}{\multirow[t]{6}{*}{\begin{tabular}{l}
MICROP ?GT \\
(. <exp GET⿰ \\
IF AA OBB OCC O ELSE HH (LC ) THEN \\
END.MICROP
\end{tabular}}} \\
\hline & \\
\hline & \\
\hline & \\
\hline & \\
\hline & \\
\hline
\end{tabular}

This could be used like:

\section*{AA 19 \$}
<variable.name> 7GT 1024 \$

Finally, microps have macro capabilities in that they can be nested end may even create several lines of code in one call (as mey be needed in a test and branch, or jump subatitute routine).

MICROP EX3
LC \(100>\)
IF EXI \$
LC ?GT 1000 \$
ELSE M 0 \$
CC 0 KNㅓ \(0 \$\)

\section*{THEN}

END. MICROP

The last major feature of the micro assembler concerns output formatting. This has not been developed at all but will consist of a basic instruction set for programmers to use to define specific output formats (i.e., hex, insertion of apecial delimiting characters, etc.). The programmer will define a function (similar to a microp or colon definition) for each type of output format. The executable code field address of the current formatting function is stored along with the other instruction word information on the first pass. On the second pass the formatting function will be executed to produce the desired result. It will be possible to change the current format function between instruction words by using a command of the form:

\section*{SET.FORMAT <format function name>}
allowing multiple output formats within a single program. By installing different formats in currently existing ones, it will be possible to view the code in punched card format as well as a format suitable for blowing PROMs!

\section*{Inplementing Techniques}

The first problem that I addressed was how to align the fields in an inatruction word definition．For words that are 32 or fower bits wide the solution is gimple， marely do logical shifting and ORing． Since 32 bits is a rather atringent limit on the word width，I have kept the same basic strategy but have defined a set of func－ tions which can do logical operations upon extended precision words．The precision （in terms of 16 －bit words）is stored in a variable called PRECISION and is set at the PROGRAM WIDTH statement．These are the extended precision functions which I needed：

1．EXT．PREC－This is a defining word that creates an extended precision variable which uses the Bartholdi＂TO concept＂to store and fetch extended precision numbers．EXT．PREC expects the desired precision of the new variable on the stack．

2．E．FILL－E．FILL expects a number and the precision of that number in terms of 16 －bit words on the steck．It uses this to fill in the most significant places with zeros until the number has a precision equal to the current value of PRECISION．Notice that the value of PRECISION must be larger or equal to the length of the given number．

3．E．DROP－This function drops an extended precision number from the top of the stack．

4．ESL－The ESL function performs a logical ahift to the left on an extended precision number．It expects the extended precision number and the number of ahifts on the stack and returna the wifted number．

5．EOR－This takes two extended precision numbers off of the stack，logically ORs them togeth－ or and returns the rosulting number．

6．EXOR－This executes an exclu－ sive OR operation between two extended precision numbers．It expects two extended precision numbers and retums the result．

7．ECOM－ECOM doe a l＇s comple－ ment of the given extended preci－ sion number．

One extended arithmetic function will be needed to implement forward referencing of labels．This function has already been mentioned and will be called E＋．





GEC 19－JUN－81） （ current address） （current field） （currerit forest） （ current instruction word） fielo leneth （field osition） （ location counter （ instruction width） （ last fatlo ） －last last format
（ current eenory addr for print routines，
（pias set at start of new instr．Mord， （ offset of shift（used in ESL），


（ extended precision Punctions GEC 12－JUN－81 ）
：EXT．PREC（＜ppecision）－र〉 builos an extended precision ） SBUILDS DUP 2草，0 D0 0 ：L00F
DOES（ \(\leqslant \boldsymbol{y} \boldsymbol{<}\) low－order．．．hish－opder，or reversed if 210 DUP DUF \(+2+\) SWAF \(_{2}{ }^{\circ}\) ZTO IFDOI ： 2 ＋100p O XTO ！ ELSE S
 PRECISION SUAP 2duf \(>\) IF DO O LOOP ELSE 2JKOF THEN；
－EDROP（＜low－order ．．．hish－order＞－く〉 dpops ext．precision ） PRECISION O DO DROP LDOF

When a field is assigned a value and is aligned, the following process occurs. An extended precision number with a precision equal to PRECISION is on the stack. This is the value of the current line of microcode. After the field-name is typed, an extended precision number with a precision equal to the width of the field is accepted. E.FILL is used on this number to make it the same precision as the instruction word, ESL is used to ahift it over the proper number of bits, and EOR is used to update the micro-instruction. This is repeated until a " \(\$\) " is encountered which will clear the flags, set any defaulted fields, store the extended precision instruction word in memory and leave an extended precision number equal to zero on the stack (for the next microinstruction).

The second main problem that I faced dealt with how to handle multiple formats. I implemented a tree structure where the instruction is the root with the hast of fields as its children. Each field רas a list of formats or a zero for its Ehildren. Every format has a list of fields as its children and the cycle continues. Each node in this tree has pointers to its parent, "oldest" child, and next youngest brather. Each node also contains a flag Jenoting whether it is a valid field or not, a value corresponding to its starting posiiton in the instruction word, its field length and its default value. Thus when a field is accessed a test is executed to determine whether it is valid or not. This s accomplished by traversing up the tree and checking the validity flag. If the first set flag is found in a field, then the programmer is trying to overwrite another format in the same field. If no flaq is set and this is not a new line of microcode, ihen this field is not defined in the same :nstruction word as the previous one(s) and another error condition is found. If, however, the field is determined to be valid, then the flag bit of that fieid will be set along with the flag of its parent, and its sarent, continuing up to the root. When a " \(\$\) " is encountered, the tree is traversed in :he same manner but from the root down and all flags are reset. At the same time any unused brothers of the lowest level 'ields used will be assigned their default values.
:NSTRUCTION FORMAT FIELD
INSTRUCTION FORMAT FIELD


( extended prec. functions - 2 GEC 12-JUN-81)
: ESL: \& low-ord... hish-ord -shifts,-clow-ord... hish, ord: shifts i-shifts to left \{drops hish ov shifts in u's \} - TO OUFLG HERE PRECISION 2 \& DUF TO PLACE HERE


TSMIFT 16 -TO TSHIFT
1 TO OUFLG
(set overflow plas)


( Extended prec. functions - 3 6EC 12-JUN-81)

OFFSET 2 t HERE + DUP \(P\) ( handies is that are splat, ROT 16 TSHIFT - - LL OR SHAF ! (into 2 bytes by shapt) THEN OUFLG NOT O TO OUFLG
-2 +LOOF DROP
PLACE HERE DO
: PPRECISION ([tof bits]m-[t of 10-bit words]) 017 H/MOD DROP SUAF EROP 14 i
 DUP FRECISION 1 - 2 t +10 I \(e^{-2}\) +LOOP

\{extended prec, functions - 4 GEC 15-JUN-81 \}
 EOR PR' HERE FRECISION
1 FRECISION 10

FRECISION WO
FRECISION \({ }^{1}+\) FICK OR - \(-1+L O O F\)
NERE FLACE \(2 O\) OL
PRECISION 2\% 0 HO IIROP LOOF
FLACE HEKE [1O 1 P 2 +LOOP:
 HERE FRECISIGN 2* +1 - DUF TO FLACE HERE
SWAF BO I ! - \(2+100 \mathrm{~F}\)
FLACE HEFE IO I E COM 2 +L00F;
: ERROR.FUNCT • ERROR CODE: * CR ;

(Extended prec, functions - 5 GEC \(15-J U N-81\) )

 1 FRECISION IO

I PRECISION + PRECISION I - + FICK
HERERFLASION 1 + PICK XOR - 1 +LOOP
PRECISION 2t O NO DKOF LOOF
PLACE HERE DO 1 E 2 tLOOF:

(offsets in field structure GEC J-JUL-81)
+ ZTOE IF ! 0 \%TO : ELSE DUF Oく, IF \(P\) THEN THEN ;
TPARENT O OFF,VAL; : PBROTHER 2 OFF, VAL ;
TANCESTOR TPARENT TPAKENT :
?INSTRUCTION. HIITH E DFF. VAL;
?FIELI.START C.FIELI B OFF.VAL ;
?FIELD. START C.FIELI B OFF.VAL ;
TFIELD.LENGTH C,FIELD 10 OFF.VAL ;
TDEFAULT C.FIELD \(12+\) i
\(\left.\begin{array}{c}\text { ( IMSTEUCIIGN ; } \\ \text { ( FIELU } \\ \text { ( FIEL } \\ \text { FIELI }\end{array}\right)\)
TDEF AULT
NEW. SON
DUP ?CHILD DUF ROT ANI
IF O SUAF EEGIN DUF PBROTHER ROT IROP DUP NOT END DROP ELSE DROP 0
THEN TO EROTHER ;

With the atructures defined, the task of creating a program comes to light. An explanation has already been given describing how the words are constructed. The following diagram should help clarify how a "program" is actually stored in memory in its first pass form.


Each program has a unique name which defines a FORTH header. When this name is typed, the program is listed in a basic binary and hex form along with the format address, LC, and any unresolved labels.

One of the primary objectiven of this micro assembler is to make microcoding easier by making it more readable, and there are quite a few places where the reverse polish notation found in FORTH does not appear quite as nice as an infix or prefix form. Hence, I have written a few short functions to allow FORTH function to accept rumbers and expressions from the input bufter as well as from the parsmeter stack

This method uses the return stack vis a function GET\# which accept input from the input buffer. If the input is a number GET places it on the stack and returns. If the input is not a number then GET* assumes that the programmer typed a left parentheses "(" meaning that there is an expression or a variable in the input buffer. If this is the case then GET will swap the last two values on the return stack and return. When a right parentheses is found, the top two values of the return stack are again swapped and the system is back to normal. This is simple and fast, although it has no method of chacking whether a set of parentheses is properly closed. However, a variable could be used which would be incremented

( headers of pields formats GEC 3-jul-81)
?MAME UUP OC, IF CFA TMAME ELSE UROP THEN ;
IGNORE 32 UORD DROF ;
: HEADER (creates lst 4 fields in FIELD and FORMAT O TO UNCLE HERE TO SELF
BROTHER OS IF SELF GROTHER TO TEROTHER
ELSE SELF FARENT TO ?CNILI
PARENT, 0, 0, 0, ( parent/orother/flas/child)
: FORMAT. HEADER ( defines FORMAT felatives i executes HEALER ) IMSTALL L.FIELD IN UNCLE INSTALL C,FIELD IN FARENT INSTALL L.FORH IN GROTMER INSTALL C.FOKM IN SELF C.FIELU NEW.SON HEADER O TO C.FIELD

(instruction and foreat defs. GEC 3-jUL-81)
: INSTRUCTION (INSTRUCTION <name; MIDTM (width;) O TO C.FIELD FORMAT. HEALIEK
GNORE GETS

UF TO F.LENGTH Jinstruction wioth
: FORMAT P-LENGTH TO F.POS i ( faeld lensth/field position
?FIELD.LENGTH TOF.LENGTM ( field lensth
PFIELD.STARTF.LENGTH + TOF.POS ( field position
FORMAT. HEADER ;

TO ZFLAG
- ZFLAG

GEGIN TFARENT XFLAG TO OUER TFLAE DUP NOT ENB DROP
YFLAG C.FIELD TO ?FLAG

```

Poraet.end and field header GEC 3-JUL-81)
FORHAT,ENL (END.FORMAT )
C.FIELIT PANCESTOR DUP TO L.FIELD TO C.FIELD
C.FIELL PPARENY IF TFIELD.START ELSE O THEN F.POS<<
IF 2 ERROR,FUNCT RESTART
ELSE TFIELD.LENGTH TO F.LENGTM
THEN ;
: FIELD.HEADER
INSTALL L.FORA IN UNCLE INSTALL C.FORM IN FARENT INSTALL L,FIELD IN BRDTHER INSTALL C.FIELD IN SELF SELF OC) IF SELF ?PARENT ELSE C.FORK THEN IUP TO PAREMT NEW.SON
HEADER:

```

erfor checking for used fields GEC 3-JUL-31 )
ER,CHECK ( check to see if field 15 Peraitted, OTO FLD.FF C.FIELD EEGIN DUF ?FLAG TO TEST.FLAG ( set TEST.FLAG=FLAG) FPDPF DROF (flip field.flio.flot ; DUP NOT TEST.FLAG OR ( if Plas Pound or io to parent ) END DROP TEST,FLAG OR FEST.FLAG FLD.FF AND EST.FLAG FLD,FF AND
IF ERROR,FUNCT KESTART ( TIELd defined twice ) ELSE TEST,FLAG NOT RESTART ( not Proper instruction) THEN O TO TEST.FLAG :
\begin{tabular}{|c|c|}
\hline  &  \\
\hline defaults & 6EC 8-JUL-81) \\
\hline \multicolumn{2}{|l|}{: DO, DEFAULT} \\
\hline \multicolumn{2}{|l|}{TFIELD.LENGTH PPRECISION} \\
\hline \multicolumn{2}{|l|}{XDEF \(\ll 2= \pm\) DROP 0 DO 0 , LOOP 0 TO XDEF} \\
\hline \multicolumn{2}{|l|}{\multirow[t]{2}{*}{\(\ll 1=3\) DROF E, FILL TDEFAULY TFIELD.LENGTH}} \\
\hline & \\
\hline \multicolumn{2}{|l|}{\multirow[t]{2}{*}{}} \\
\hline & \\
\hline \multicolumn{2}{|l|}{} \\
\hline TO. DEFPSELTO'ZDEF ; & : IMIT. UEF 2 TO XDEF ; \\
\hline \begin{tabular}{l}
default \\
GET 4 TO.DEF DO. DEFAULT ;
\end{tabular} & \\
\hline
\end{tabular}
when \(a\)＂ n ＂is encountered and decrement－ ed when a＂）＂is found．This would catch

\section*{\(\because: \mathrm{ETH}\) DEFINITIONS}

A typical usage of this function could こe：
\[
:\{+\} \text { GET } \#+\text {; }
\]
\[
3\{+\}(4\{+\} 5) .
\]

12
\begin{tabular}{|c|c|c|c|c|}
\hline \[
\begin{aligned}
& \text { orrent } \\
& \text { inction } \\
& \hline
\end{aligned}
\] & Command & Parameter Stack & Return Stack & \\
\hline \multicolumn{5}{|l|}{} \\
\hline main & 3 & 3 & － & input a 3 \\
\hline （－） & （＋） & 3 & main & call function \((+)\) \\
\hline SET\＃ & GET\＃ & 3 & main \(\{+\}\) & call function GET\＃ \\
\hline & （ & 3 & （ + ）main & swap return stack \\
\hline main & 4 & 34 & （＋） & return and input a 4 \\
\hline ：＋） & \｛＋\} & 34 & \((+)\) main & call（ + ）again \\
\hline 3ET\＃ & 5 & 345 & \｛ + \} main ( + ） & input a 5 \\
\hline ！+ & ＋ & 39 & \((+)\) main & return and add \\
\hline main & & 39 & \｛＋\} & return to main \\
\hline ） & ） & 39 & （t）main & call function） \\
\hline & & 39 & main \({ }^{\text {＋}}\) ） & swap return steck \\
\hline （＋） & ＋ & 12 & main & return and add \\
\hline nain & ． & － & － & return and print \\
\hline
\end{tabular}

There are a few general concepts mich are used throughout this micro assembler，one of which is the＂TO con－ ：ept＂（see Joe Sawicki＇s paper entitled EDtimized Data Structures for Hardware Eontrol）．This concept allows the use of －ariables without the programmer having ：o deal directly with the address．While ：his may be thought of as being a bit un－
any errors involving too many closing par－ entheses．A＂］＂function could be written which would behave in the same manner as ：he UCl LISP function of the same name． ＇：would use the variable mentioned above is close all open parentheses for a suc－ zessful evaluation of the expresaion．

GET\＃and its related algebraic func－ ：ons have some interesting features in ：nat there is no hierarchial ordering of ＇jnctions（i．e．， \(2+3 \bullet 5=25\) while 5© \(3+\) \(:=17\) ），however，expressions enclosed in zarentheses will be solved before others ．e．， \(2+(3-5)=17)\) ．The entire code for \(: \mathrm{in}\) is only a few lines long and is as ＇3：10ws：

SET： 32 WORD NLMBER
NOT IF R＞R＞SW＇AP \(>R>R\) THEN ；

ここABCLLARY ALGEBRAIC ALGEBRAIC DEFINITIONS redefine functions
```

```
* GET# + ; 首 - GET# - ; rem ; re-swap return stack
```

```
* GET# + ; 首 - GET# - ; rem ; re-swap return stack
    R>R> Sh'AP >R >R ; swap return stack
```

    R>R> Sh'AP >R >R ; swap return stack
    ``` swap return stack
```


gets number
swap if not a number

FORTH－like，it does result in much cleaner code．I adapted the concept in one place to build a flip－flop function． This function creates a data type which alternately returns zeros and ones when－ ever it is called and makes use of the＂TO concept＂to allow itself to be initialized to either state．The micro assembler also makes use of multiple vocabularies to allow the same function to have different meanings in different contexts．While this is not absolutely assential for the assem－ bler to run，it again makes the code cleaner and easier to use．

## Concluaion

The reason why I have chosen to write this micro assembler in FORTH is simpli－ city．As 1 mentioned earlier，this＂pro－ gram＂is based largely upon a very lengthy micro assembler written by Signetics and yet the FORTH code is only a few pages long．The time spent programming was equally short．It took roughly half of my time at work from around June 10 through July 15 to complete the micro assembler to this point（although I have occasionally gone back to add or change a feature or two）．Two of the features that I did change，labels and forward referencing through the first pass，brought up another quality of FORTH：its modular nature． These are rather major additions and yet they only required one new＂block＂of code，a few minor changes in the old code and took only a few hours to implement［

Once the forward referencing is com－ pleted and the output formatting is imple－ mented，this code will be a micro assem－ bler by itaelf as well as a kernel for more extended versions．An example of an extended feature is the compilation of a symbol table at the end of a program．A further extension would involve tying this symbol table to ather symbol tables to allow external references．Through the use of external symbol tables the micro－ code could be maintained in the first pass format so that the external references could be resolved several times for labels with differing values．This could result in a modular microcoding technique． Another extension could be a FORTH pro－
gram which would be used, in much the same manner as the micro assembler, and similar to Hardware Description Languages, to describe simulator for the microcade. These two programs would constitute a powerful yet inexpensive teaching aid as well as an effective design tool. Programmers and students would not need to waste their time punching cards or blowing PROMs in order to discover the errors in their codel A dozen other "nice" features can be imagined (i.e., prohibiting forward referencing to allow interactive microcoding, or the development of intrinsic microps to define commercial chips, etc.), but the point is that they could all be based around the small "kernel" micro assembler presented here.

## Acknowlodgements

I would like to thank Lawrence Foraley for the time and effort he expended helping to direct and complete this project. I would also like to extend thenks to Dr. Charles Merriam for his useful comments and suggestions.

This work was partially supported by the following sponsors: Exxon Research and Engineering Company, General Electric Company, New York State Energy Research and Development Authority, Northeast Utilities, The Standard Oil Company (Ohio), the University of Rochester, Empire State Electric Energy Research Corporation, the Center for Naval Analysis, under grant number CNA SUB N00014-76-C-0001 and the U.S. Department of Energy inertial fusion program under contract number DE-AC0B$800 P 40124$.
G.E. Cholmondeley is currently an undergraduate student in the department of Electrical Engineering at the University of Rochester. His interests lie in computer software and hardware design.

1. Signetics Micro Assembler Reference Manual

## HELP WANTED

## FORTH Software Enginoer

Program, edit and maintain files for 8080. Ability to troubleshoot the software-hardware interface.

## Call:

Wendy Palmer
1-800-225-4040
Instrumentation Laboratory, Inc.
Analytical Instrument Division Jonapin Road
Wilmington, MA 01887

: U. ZEKO
IUP 4096 T U:= IF OT
UUF 256T U〉z IF 17
ELSE QUF 16T U $\mathrm{ELSE}=1 F 2 T$
THEN THEN THEN
4 SWAF - U.R ;


```
    *Pintins routines - 2 GEC 16-JUN-81)
: #PRINT ( <ext.pre.t.addr>-<> print ext.preit 1r, binary & hex )
    IUF'FFECISION 2T + SUAF 2DUP UOI B. 2T +LOOP
        .TO DO I U.ZERO 2T +LOOP:
: MEM.INC MEM DUF 2+ TO MEK P ;
```



```
1 pririting routames - 3
GEC
16-JUN-81)
1.FASS.FRINT
    UUFF TO, MEM O I AND
        IF EGKROR PENOGAM LENGTHOC CR
            IF BEGIN
                    :LABEL : MEM DUP CFA TMAME CR 2+ 10 MEM
                    ANSHIFTEI: - MEH DUP E PFR CR 2+ DUF
                    END
            THEN MEM 2+ TO MEM
        \because FDRNAT: MEN DUP O CR 2+ IO MEM
```



```
        CFH PK IN CR MEH FRECISSON 2I + }10\mathrm{ MEH
        THEN;
\begin{tabular}{|c|c|c|c|}
\hline  & BLOCK & 183 &  \\
\hline prosram statement & & GEC & 16-JUM-81) \\
\hline
\end{tabular}
: FROGFAH
    RUILSS IGNOFE GET* DUF , ?PRECISION TO PRECISION O ,
        1 TO NEW. WOKII
    OL E.F1!L
    DOES: [UFF PPRECISION TO PRECISION 4 + 1.PASS.PRINT ;--*
```



```
    end proyram & Hacrof commands GEC
    ENL.PFOGRAM
        ETIROF1:
    MICROF [COMFILE] : ;
: ENII.hICROF [COMFILE] ; ; IMNEDIATE
    SET i defines a variable data type)
    HULLLIS IGNORE GETH , (SET \var,name% TO (expression),
    ( <var,name) returns value)
: EQU ( EQU <var; name` WITH <expression>)
    ]'[ IGNORE GETG SHAP!;
MICRO DEFINITIONS

\section*{NDUSTRY NEWS}

\section*{FORTH-Based Savvy Lets User} Talk to Computor

FOR TH, Inc. is working with its parent company, Technology Industries, Inc. of Sente Clara, Cellfornie, to develop a new eoftware package for the Apple II, using a 280 processor. With it, the Apple will offer the kind of casual and efficient mancomputer interface that until now, existed only in movies like 2001 and Star Wers.

The project calle for Savvy--the trade name for Excalibur Technology Corporation's Adaptive Pattern Recognition Pro-cessor--to be used as a unique language interpreter. Savvy permits a user to communicate with a computer in the user's native language and normal praseology-no special language and formm are needed. Specifically, Savvy:
- Recognizes written words strung together in idiomatic phraces. (Future vertions will understand spoken words and respond to Spenish commands as well as English. Other lenguages will follow.)
- Translates these imprecise patterns into precise computer cormmande.

Savvy's unique interactive approach to dealing with computers is an important development for the 80s. The powerful combination of FORTH and Savvy will be significant in realizing the aystem's full potential and demonatrating the power of FORTH. A special development team has been formed for this project, including Art Gravina, Chuck More, Dean Senderson, and another programmer who has not been identified.

NO ROOM FOR THE ORDER FORM THIS TIME!
ORDER - Proceedings 1981 Rochester PORTH Standards Conference. Send check or MO to FIG in US funds on US bank, \(\$ 25.00\) US, \(\$ 35.00\) Foreign.



\section*{Published by Forth Interest Group}

Volume III No. 5
Prolisher
Editor
Editorial Review Board
Bill Regadale
Dave Boulton
Kim Herris
John James
Dave Kilbridge
Henry Laxen
George Maverick
Bob Smith
John Bumgarner
FORTH DIMENSIONS solicits editorial material, commente and letters. No responsibility is aseumed for accurecy of material submitted. ALL MATERIAL PUBLISHED BY THE FORTH INTEREST GROUP IS \(\mathbb{N}\) THE PUBLIC DOMAIN. Information in FORTH DIMENSIONS may be reproduced with credit given to the author and the Forth Interest Group.

Subscription to FOR TH DIMENSIONS is free with memberahip in the Forth Interest Group at \(\$ 15.00\) per year ( \(\$ 27.00\) forsign air). For membership, change of address and/or to aubmit material, the address is:

\section*{Forth Interest Group}
P.O. Box 1105

San Carlos, CA 94070

\section*{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 availsble programming tools, especially for observatory automation.

Mr. Moore and several aseociates formed FORTH, Inc. in 1973 fo the purpose of licensing and support of the FORTH Operating System and Programming Language, and to aupply application programming to meet customers' unique requirementa.

The Forth Interest Group is centered in Northem Californic. Our membership is over \(\mathbf{2 , 4 0 0}\) worldwide. It wes farmed in 1978 by FORTH programmers to encourage use of the language by the interchange of ideas through seminars and publications.

\section*{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 heip and support I have received from our readers. I have not done everything right and some of the best help has been your 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 erticles designed to help our entry level readers. This, however, will not be done at the expense of our more seasoned FIGGERS who will find en 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, FOR TH DIMENSIONS is what you make it.

\section*{C. J. Street}

\section*{PUBLISHER'S COLUNN}

1981 has been a great year for FORTH, the FORTH Interest Group and for me, personally. FORTH has spreac 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 eurprise. I stand in good company.

Roy C. Martens
tions of voluntears sarving without reimbursement. The FORTH DIMENSIONS editorial staff supports FIG efforts to keep FORTH intact and resist the temptation to obtain mere popularity and in the process, fail in their mission to provide and support the finest software concepte and tools available today. This has not been an easy task (and all too often, a thankless one) but it is hoped that if others will least try to understand, the efforts and contributions of these volunteers will continue to benefit us all.--ed.

\section*{Dear Fig:}

Congratulations to all the people who produce FORTH DIMENSIONS on its quality and improvement. Plesse 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 elee?
--ed.

\section*{Dear Fig:}

Glen Haydon's nice article in FORTH DIMENSIONS III/2, page 47 talks about an algorithm he would like to have to Jetermine the Julian day. With the sackground that FORTH has in astronomy, :'m sure there must be several, but this is :he nicest I know. It comes from the U. S. Vaval Observatory via an article in the Astrophysical Journal Supplement Series, vol. 41 No. 3 Nov. 1979 pp 391-2.
( JULIAN DATE)
: JD >R SWAP
DUP \(9+12 / R+7\) * \(4 /\) MNUS
OVER 9-7/R+100/1+3*4/-
SWAP 2759 */ +
+ S-> D \(1.721029 \mathrm{D}_{+}\)
367 R) M* D+;

Example: 3201982 JD D. 2445049 OK

If you are only concerned with dates Jetween \(3 / 1 / 1900\) and \(2 / 28 / 2000\), then you zan amit line 3 entirely.

On another subject, there is enother zorrection 1 noticed in the dump of the fig-FORTH 6502 Assembly Source - at xation OC32, 80 lA should be D7 OB.

Peter 8. Dunckel
52 Seventh Avenue
San Francisco, CA 94118
Really slick! But the algorithm would be ard to explain to most people.--ed.

\title{
FUNCTIONAL PROCRANNING AND FORTH
}

\author{
Harvey Glees \\ Univeralty of South Floride \\ College of Engineering \\ Department of Computer Science Tampa, FL 33620
}

The diatingulshed computer scientist, John Backus, in his 1977 Turing Award lecture (1) describes the shorteomings of conventional programming languages and suggeste a new epproach to programming in a style deecribed as functional programming (FP). We will summarize the faults that Beckus finds in conventionsl languages, briafly dascribe the functional programming style, and lastly show that FORTH moets the spirit of this style of programming.

\section*{Conventional Lanquages}

An underlying problem of conventional programming languages is that they tend to be high level descriptions of the Von Noumann computer. The assignment statement is the principal construct of these languages. A program becomes a series of these assignment statementa, each of which requires the madification of a single cell. We may think of the Von Neumann computer as a set of storage colls, asparate 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 Noumam design. Conventional languages in the "word at a time" flow described above require large data transfers through this amall 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. Backus refers to it as an "intellectual bottleneck." He characterizes conventional languages as both fat and weak since increases in the size and complexity of these languages have provided only small increases in power. The typical programming language requires a large fixed set of constructs, is inflexible, and is not extensible. The problem has been eased by approaches such as top-down design and structured programming, but these have not provided a solution to the underlying difficulty. Backus auggests that we need a new way of thinking about computing. He describes a new style which he calls functional programming.

\section*{Functional Programming}

This new style of programming has the following characteristics:
- A function (program) is constructed from a set of previously defined
function using a set of functional forms that combine these existing functions to form new ones.
- The most fundamental functional form is called composition. If the composition operator is denoted by 0 , then in Backus' notation "fog" is the function where \(g\) is first applied and then \(f\).
- The functions incorporate no data and do not name their conventions nor substitution rules.
- A function is hierarchical; i.e., built from simpler functions.

Backus points out that, "FP (Functional Programming) systems are so minimal that some reader 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 construct denoting functions which take function as parameters. For example, the construct "if-else-then", and the construct "do while" are functional forms. As indicated above, composition is slso 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 functianal forms is not inherent in FP systems. Backus defines a language with this capability as a formal functional programming (FFP) lenguage.

\section*{An Example of Functional Programming:}

\section*{The Factorial Function}

An example of a program written in the style of functional programming is as follows:
def \(!\equiv\) eq \(0 \rightarrow \overline{\mathfrak{l}} ;{ }^{\circ}\) o [id, \(:\) s subl], where the notation \(o, \equiv\), and [] denote functional forms. As we have seen, o denotes composition. The notation \(\left[f_{1}, f_{2}\right]\) denotes construction where \(\left[f_{1}, f_{2}\right]\) applied to en argument \(x\) yields the sequence \(\left\langle f_{1}(x)\right.\),\(f_{2}(x)>\). The notation \(p \rightarrow f ; g\) 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 useu in the above are:
eq0 applied to \(x\) yields a value true if \(x\) is 0 , and yielda false otherwise.
\(\overline{1}\) is the literal value 1 and yields the
value 1 , regardiess of the argument.
- is the multiplication operator, and applied to a sequence \(\langle x, y\rangle\) yielde \(x^{*} y\).
id is the identity operator. id applied to \(x\) yielde \(x\).
subl applied to an argument \(\times\) yields \(x\) 1.

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

Clearly then for \(n>0\) this is a definition of the factorial function. In FORTH (if the language were recuraive) we would write:
```

: !
DUP 0= IF 1+
ELSE OUP 1.-n.
THEN ;

```

The syntaxes of the two examplea are different. The composition rule is applied right to left in the first example and left to right in FORTH. The rules for dropping arguments are different. Construction is not used in FORTH." That the rules of syntax are different should not be surprising. The operations were defined by different people at different times. What is most important is that on close examination it is epparent that the style is essentially the same. We have "words" which denote functions which are evaluated following very similar rules.

FORTH as a Lanquage with Characteristics of Functional Progremming

Consider the FORTH (outer) interpreter. Literally all that the interpreter recognizes are functions; or to be precise, words that denote functions.** The fundamental combining form is composition where in FORTH "fog" would be expressed as \(g\) f. Functions need not incorporate data, do not name their arguments, and require no substitution rules for parameter passing. There are no assignment statements and a new function is built from simpler previously defined functions. It is this style of programming in FORTH-so different then that of conventional lang-uages--that provides a power and flexibility that has sparked the enthusiasm of so many of us.

\section*{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-
larly among those interested in innovative approaches to computer architectures. It is this author's contention that FORTH is a functional programming language which closely resembles the approach suggested by John Backus in his definitive paper. It will be interesting to see if , as a reault of this paper, languages which have attributes similar to FORTH begin to appear in academic circles.
- The suthor has recently implemented such an operator in FORTH.
** The way that litersls are handled can be viewed as merely a question of implementation and efficiency.

\section*{References}
1. J. Backus, "Can Programming be Liberated from the Von Neumann Style?" CACM, Vol. 21, No. 8, Auguat 1978, p. 613.

\section*{FORTH AND ARTFICIAL LINGUISTICS}

\section*{Reymond Weisling \\ Surakarta, Jawa Tengah \\ Republik Indonesia}

Thare has not been much said about the linguistic nature of computer languages, principally because so few of them permit the development of syntax structures that apporach human lenguage, and hence foster linguistic observation. FORTH and its other threaded-code relatives allow for such structures to be developed, principally because of the larger body of words that arise from its extensibillity and hierarchal function of operators.

The point I wish to address here is the syntactical limitations of the language we are building, an artificial lenguage based in part on a tuman Ienguage (English) that is widely used wherever technology has developed. But there is a fundamental weakness in this English which I think we must be aware of, since it runs counter to the philosophy of FORTH. This is the syn-tax-senaitivity of word forms, especially noune and verbs, which in Englith are commonly apelled and pronounced exactly the same. We rely on the atructure (wordorder, partly) to distinguish these often unrelated words.

A few examples are in order. Consider the possible function of these FORTH words, both with respect to their current use (some are nouns while others are verbs), but also in their opposite hypothetical use: BUFFER , FENCE , KEY , LIMATT, LOOP , SPACE ,TYPE , etc. Others which a programmer might wish to use in developing applications might include: OFFSET, SPAN , INSERT,

FILE , CATALOG , OUTPUT. . Since the action of these words is not known from the word itself, but only from either previous agreement or syntax, and since syntax sensitivity is not a common part of FORTH (i.e., where a syntactical form does not alter the way in which a word is compiled), some degree of confusion can result.

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 verDs) cannot function as nouns despite 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 samenamed word to use based on its syntactical position, like nouns (variables, constants, arrays) being objects of TO and FROM ? Or does this push us back into the horrible mess of artificial syntax forms such as algebraic notation (something we are perhaps proud to have departed from)?

I offer no solution per se. I only wish to point out a weakness that we all should be sensitive to when we assign names to our words. Since FORTH is still in evolution, this is yet another aspect to consider when standards are defined. I wish to disclaim any implication that I am a linguist of any sort other than Armehair Linguist. My sensitivity to this is a result of living in a different culture where I am learning a human language that permits far greater fluidity of structure due to the inherent differences in nouns and verbs, shown by a well codified system of prefixes and suffixes (morphemological differentiators). Those here who learn English struggle with the structural differentiation of all the parts of speech while our morpheme differentiators are used for relatively useless things like verb conjugation, plurality, cases, and tenses (which are all essentially absent in this part of the world). As technology spreads, an artificial language for man-machine manipulation (a two way street) should be more universally based, at least with respect to linguistic modeling. As FORTH is already in use in many parts of the world, the channel for feedback is already open.

\section*{FORTH STANDARDS CORNER}

Robert L. Smith

\section*{More Words on WORD}

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

\section*{: WORD WORD HERE;}

Jther implementers would probably put :ne string elsewhere. Now suppose that : he user wished to reverse the character s:ring and emplace the modified string in : e dictionary. The result from the for--er implementer's system will not be as expected, and will not result in "equivaent execution" on the later implementer's irstem. A similar but much less serious =-oblem occurs with PAD. PAD is - onventionally affset from HERE by a - xed amount ( 68 bytes in fig-FORTH). -here are at least three different soluons:
(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.

\section*{-et Me Number the Ways}

In many areas the 79-Standard defines nits and formats in painful detail. There \(s\) an important area in which very little is sid, namely the format for single and zouble precision numbers in the input s:ream. In the section "interpreter, text"
is clear that "numbers" are allowed in :he input text stream and may either be ampiled or placed on the parameter s:ack. A definition of the format of \(a\) number should include at a minimum the : stinction between double and single pre: sion, the sign of the number, and the set :: allowed characters from which the umber is constructed. In keeping with : רe spirit of the rest of the Standard, I -ould like to propose a few definitions - 7ich should be fairly easy to implement Ind which appear to be compatible with most current implementations (including

\section*{fig-FORTH). First, we define aligit!}

\section*{digit}

A diglt 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 D ... 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:

\section*{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.

1 recommend that implementers allow the sbove 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.

\section*{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 (415) 538-3580. See back issues of FORTH OIMENSIONS for further information, or just call up and send a few carriage returns until the system responds.

\section*{CORRECTIONS}

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

\section*{REFERENCES}
1. Forsley, Lawrence P. The Beta Laser Control System. A talk given at the Laboratory for Laser Energetics on March 9, 1977 and on July 16, 1977 at the Wilson Synchrotron, Cornell University.
2. Forsley, Lawrence P. "F orth Multitasking in URTH". The Best of the Computer Faires volume IV. San Francisco: 1979.
3. Boles, J. A., Pessel, D. and L. P. Forsley. "Omega Automated Laser

Control and Data Acquisition". IEEE Journal of Quantum Electronics, Vol QE-17 No. 9. New York, Now York: IEEE, September, 1981.
4. -an-.: Towsids More Usable Syotems: The LSRAD Report. Application Development). Chicago: Share, Inc., 1979.
5. …... IEEE Standard 583-1975. Now Yorki IEEE, 1975.
6. -----. 1977 Lsboratory for Laser Energotics Annual Report. Rochester, NY: Laboratory for Laser Energetics, 1978.
7. Moore, Charles. "Forth: A New Way to Program Minicoimputers" Journal of Astronomy and Astrophysics Supplement 15. New York: AAAS, September, 1974.
8. Moore, Charles. "Forth, The Past Ten Years, and the Next Two Weeks". Forth Dimensions. Vol. 16 San Carlos, CA: Forth Interest Group, 1979.
9. Rather, Elizabeth and Charles Moore. "The FORTH Approach to Operating Systems". ACM 76 Proceedings. New York: ACM, October, 1976.
10. Ritchie, D. M. and K. Thompson. "The UNIX Time-Sharing System". The Bell System Technical Journal. Vol. 57 No. 6 Part 2. New Providence, NJ: A.T. and T., July-August, 1978.
11. Ritchie, D. M., et al. "The C Programming Language". The Bell System Technical Journal. Vol 57 No. 6 Part 2. New Providence, NJ: A.T. and T., July-August, 1978.

Change: FDIII/4, pg. 118, para 3 to: The TO concept was developed by Or. Paul Bartholdi \({ }^{2}\) as an alternative to constants and variables.

\section*{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!

\section*{TECHNOTES, BUGS AND FIXES}

\section*{have three questions about FORTH:}
Q. I know of two CP/M FORTH 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 figgers 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(?), Pve been leaving a lot of control characters behind when using the editor. They don't show up on a screen liat 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 charaters with spaces for anyone else who has this problem.

SCREEN: 95
(HUNT FOR CONTROL CHARACTERS)
: HUNT (SCREEN \# ---)
BLOCK
10240 DO DUP C@ DUP 32 < IF CR . \(4+164\) + EMIT
." (a) :" DUP U. ELSE DROP ENDIF 1+ LOOP DROP ;
: FIXSCREEN (SCREEN :--)
BLOCK
10240 DO DUP Ca 32 < \(F\) DUP 32 SWAP C! ENDIF 1+ LOOP DROP ;
( ACTUALLY HRNT AND FIXSCREEN ARE QUITE SIMILAR, HUNT JUST
SHOWS UP ANY GURLTY CHARACTERS ANO 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 PP 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)

\section*{THAT MYSTERIOUS fig-FORTH ANNESIA}

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 clobbering 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 64 K , it may address RAM where there are no boarde). 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 B071 E. 7th Street, \#14 Buena Park, CA 90621

\section*{TRANSIENT DEFINITIONS}

These utiliites 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 eurrent 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 figFORTH definitions.

Philip Wasson

\section*{MORE WORDS ABOUT WORD}

Robert D. Villwock
Microsystems, Inc.
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 tacitly assumed 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 :28, 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 taken buffer and PAD float above HERE or are fixed ocation 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 NORD. 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 79-5TANDARD requires that PAD be able to hold at least 54 bytes, so now we're at HERE + 320 sytes.

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 ISTACK 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 al:owed. Even if you start automatically :educing the PAD offset so that it remains fixed in size, the token buffer begins strinking and can no longer satisfy the 256 oyte 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 the ir space. The floating buffer concept seems to obecure 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 IK bytes of apace left: Since most of the time the takens 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 32 K to 48 K bytes, exclusive of the dictionary headers and the FORTH operating sysem 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 implementation which is called proFORTH \({ }^{\top M}\), 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 prof ORTH, 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 PAOs of arbitrary length becomes very incongruous. Admittedly, l've described a somewnat 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 thave your cake and eat it toon, a rather simple factoring of WORD can easily accomplich it. To illustrate my point, euppose we devies a more basic WORD called (WORD) and define it es followe:
```

|(WORD) ( c -- a n) BlK - TDUP
IF block
FLSE TIB
THEN >IN + SWAF ENCLOSE
>IN + \& OVER - -ROT + SWAP :

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

This now (WORD) extracts the next token from the text stream, delimited by c , and leaves its address and length on the stack. Actually, the token ia merely left in the input buffor (keyboard or diak) 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 s "dot-quote" string, a definition such as WORD, can be used.
- WORD, ( \(C\)-- )
(WORD) HERE OVER 1+ ALLOT SWAP OVER C1 COUNT CMOVE ;

If you want a blank-filled line put in PAD, the following could be used:
- TEXT ( c -- ) PAD C/L 2+ BLANKS
(WORD) C/L MIN PAD CI PAD
COUNT CMOVE :
For the routine compiler/interpreter job of extracting smsll ( 31 characters or less) tokens from the text stream, the following could be used:
```

| WORD ( c-. a )
(WORD) WDSZ MIN WBFR CI KBFR COUNT 1*
GMOVE WBFR ;

```
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 delimiter, or for that matter, the leading count byte for 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 fundemental token extractor allows implementors to compile dot quote strings, for example, without the need for eny 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 proF ORTH is being used, the target de finition body can optionally be compiled "in place" separate from the dictionary header. There may be room for the atring 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 buffera 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 handied by WORD.

\section*{CORRECTION TO FEDIT}

Sorry you had trouble with FEDIT. The listing was retypad at FIG and eeveral typos creeped in. They are:
1. SCR 64 Line 10: compile should be COMPILE
2. SCR 65 Line 23: 1+/MOD should be 1+ \(16 / \mathrm{MOD}\)
3. SCR 67 Line 48: 8/BUD should be B/BUF
4. SCR 67 Line 49: :e should be: .E
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 FIC about this and they were acutely aware of the problem but they are simply not set up to directly reproduce listings into FD 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 FigFORTH line editor definition, R. I used the pseudonym because FEDIT was the first progrem 1 wrote in FORTH and I resily wasn't familiar enough with Vocabularies 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 could run one for you from my current version.

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

\section*{A HELPFUL UTRITV}

Here's a short FORTH word of great utility that 1 use heavily in my screens. I hope you like it. Its name is CVD, which stends for "convert to decimal".

\section*{DECIMAL}
: CVD
BASE SWAP
OVER /MOD
ROT / MOD
\(10^{\circ}+\)
\(10^{\circ}+\)
;
I like to work in hexidecimal, but often make mistakes when using the words LOAD, LIST, and many of the FORTH acreen editor words because I'm thinking in decimal when the systern's in hex. If I do the following:

\section*{: 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 bese accepts the number.

As to how it works, a little work will show that CVD aplits 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.

\author{
Gregg Williams \\ BYTE Publications \\ PO Box 372
}

Hancock, NH 03449

\section*{CALL FOR PAPER5}

\section*{1982 Rochester FORTH Conference} on
Data Bases and Process Control
May 17 through May 21, 1982

\section*{University of Rochester \\ Rochester, Now York}

The second annual Rachester 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 FOR TH.

There is a call for papers on the following topicas
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, metacompilation, 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 chairmen, 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

\author{
A FORTH ASSEMBLER FOR THE 6502 \\ by William F. Ragadale
}

\section*{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, macroassembler 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 thia is the best FORTH assembler evar distributed. It is the only such assembler that detects all errors in op-cade generation and conditional structuring. It is released to the public domain as a defense mechanism. Three good 6502 assemblers were submitted to the FORTH Intereat 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.

\section*{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, for clarity of expression. Functions may be written first in high-level, tested, and then re-coded into assembly, with a minimum of restructuring.

\section*{THE ASSEMBLY PROCESS}

Code assembly just consists of interpreting with the ASSEMBLER vocabulary as CONTEXT. Thus, each word in the input stream will be matched according the Forth practice of searching CONTEXT first then CURRENT.

ASSEMBLER (now CONTEXT)
FORTH (chained to ASSEMBLER)
user's (CURRENT if one exits)
FORTH (chained to user's vocab)
try for literal number
else, do error abort
The above eequence is the uaual action of Forth's text interpreter, which remains in control during assembly.

During assembly of CODE definitions, Forth continues interpretation of each word encountered in the input atream (not in the compile mode). These assembler worda specify operands, address modes, and op-codes. At the conclusion of the CODE deflnition a final error check verifies correct completion by "unsmudging" the definition's name, to make it available for dictionary searches.

\section*{RUN-TIME, ASSEMBLY-TIME}

One must be careful to understand at what time a particular word definition executes. During assembly, each assembler word interpreted executes. Its function at that instent is called 'assembling' or 'assembly-time'. This function may involve op-code generation, addresa calculation, mode selection, etc.

The later execution of the generated code is called 'run-time'. This distinction is particulary important with the conditionals. At assembly time each such word (i.e., IF, UNTIL, BEGIN, etc.) itself 'runs' to produce machine code which will later execute at what is labeled 'run-time' when its named code definition is used.

\section*{AN EXAMPLE}

As a practical example, here's a simple call to the system monitor, via the NMI address vector (using 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 vocabularly.

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

\section*{OP-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 end in ",". The significance of this is:
1. The comma shows the concluaion of a logical grouping that would be one line of clasaical assembly source code.
2. "," compiles into the dictionary; thus a comma implies the point at which code is genereted.
3. The "," distingulshes op-codes from possible hex numbers ADC and ADD.

\section*{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.

\section*{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.

\section*{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 ENDCODE. Legality of address modes and operands is insured by means of a bit 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:
\[
\begin{array}{ll}
\text { i.e., } & 0=\text { of ASSEMBLER when you want } \\
0=\text { of FORTH }
\end{array}
\]
(Editor's note: the listing assumes that the figf ORTH error messages are already available in the system, as follows:
?CSP issues the error message "DEFINITION NOT FINISHED" if the stack position differs from the value saved in the user variable CSP, which is set at the creation of teh definition name.
?PAIRS issues the error message "CONDITIONALS NOT IMPAIRED" if its two argurnents do not match.

3 ERROR prints the error message "HAS INCORRECT ADDRESS MODE".)

\section*{SUMMARY}

The object code of our example is:
\begin{tabular}{ll}
305983 4D 4F CE & CODE MON \\
305D 4D 30 & link field \\
305 F 6130 & code field \\
306100 & BRK \\
30624 C 4202 & JMP NEXT
\end{tabular}

OP-CODES, revisited
The bulk of the assembler consists of dictionary entries for each op-code. The 6502 one mode op-codes are:
\begin{tabular}{llll} 
BRK, & CLC, CLD, CLI, & CLV, \\
DEX, & DEY, & NNX, & NNY, \\
PHOP, \\
RHA, PHP, PLA, PLP, & STI, \\
TAY, & TSX, & TXS, SEI, & TAX, \\
TXA, TYA,
\end{tabular}

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

The multi-mode op-codes are:
\begin{tabular}{llll} 
ADC, AND, CMP, EOR, LDA, \\
ORA, SBC, STA, ASL, DEC, \\
INC, LSR, ROA, ROR, STX, \\
CPX, CPY, LDX, LDY, STY, \\
JSR, JMP, BIT, &
\end{tabular}

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


\section*{EXAMPLES}

Here are examples of Forth vs. conventional assembler. Note that the operand cames firat, 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.
\begin{tabular}{rl} 
IA ROL, & ROL A \\
1 LDY, & LDY \#1 \\
DATA ,X STA, & STA DATA,X \\
DATA,Y CMP, & CMP DATA,Y \\
\(6 \times\) ADC, & ADC (06, X) \\
POINT )Y STA, & STA (POINT),Y \\
VECTOR ) JMP, & JMP (VECTOR)
\end{tabular}
(.A distinguishes from hex number 0 A )

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.

\section*{6502 CONVENTIONS}

\section*{Stack Addressing}

The data stack is located in z-page, uaually addressed by "Z-PAGE, \(X\) ". The stack starts near \(\$ 009 E\) and grows downward. The \(X\) index register is the data stack pointer. Thus, incrementing \(X\) by two removes a data stack value; decrementing \(\times\) 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 \(X^{\prime \prime}\) 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 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 2-page.
\begin{tabular}{|l|}
\hline \begin{tabular}{l} 
sec high \\
sec low
\end{tabular} \\
\hline \begin{tabular}{l} 
bot high \\
bot low
\end{tabular} \\
\hline
\end{tabular} \begin{tabular}{l} 
s=sX offset \\
above \(\$ 0000\)
\end{tabular}

Here is an examples of code to "or" to the accumulator four bytes on the stack:
\begin{tabular}{ll} 
BOT LDA, & LDA \((0, x)\) \\
BOT 1+ ORA, & ORA \((1, x)\) \\
SEC ORA, & ORA \((2, x)\) \\
SEC 1+ ORA, & ORA \((3, x)\)
\end{tabular}

To obtain the 14 -th byte on the stack: BOT 13 + LDA,

\section*{RETURN STACK}

The Forth Return Stack is located in the 6502 machine stack in Page 1. It starts at \$OLFE 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, PLA, which will pull the low byte, then the high byte from the return stack. To operate on aribitrary bytes, the method is:
1) save \(X\) in \(X\) SAVE
2) execute \(T S X\), to bring the \(S\) register to \(X\).
3) use RP) 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 nondestructively tests that the second item on the return stack (also the machine stack) is zero.

\section*{CODE IS-IT (zero? )}

XSAVE STX, TSX, (setup for return stack)
RP) 2+ LDA, RP) 3 + ORA, ( or 2 nd item's two bytes together)
\(0=\mathrm{IF}, \mathrm{INY}\), THEN, ( if zero, bump \(Y\) to one)

TYA, PHA, XSAVE LDX, (save low byte, rstore data stack) PUSH JMP, END-CODE ( push


\section*{FORTH REGISTERS}

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

IP address of the Interpretive Pointer, specifying the next Forth address which will be interpreted by NEXT.

W address of the pointer to the code field of the dictionary definition just interpreted by NEXT. W-1 contains \(\$ 6 \mathrm{C}\), the op-code for indirect jump. Therefore, jumping to \(\mathrm{W}-1\) will indirectly jump via W to the machine code for the definition.

UP User Pointer containing address of the base of the user area.

N a utility area in \(z\)-page from \(\mathrm{N}-1\) thru \(\mathrm{N}+7\).

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.

\section*{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:

\section*{CODE DEMO}

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

\section*{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 syatem 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, \(\mathrm{N}-1\) 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 I - STA,
        (setup a counter)
BEGIN, 8001 BIT,
    (tickle a port)
    N 1 - DEC,
        (decrement the counter)
    \(0=\) UNTIL, NEXT JMP, END-CODE
        (loop till negative)

\section*{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,


\section*{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 con-
trol words:
\[
\begin{aligned}
& \text { BEGIN, UNTIL, IF, ELSE, } \\
& \text { THEN, }
\end{aligned}
\]

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 occura: Highlevel 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,).

\section*{Examples are:}

PORT LDA, \(O=I F,\langle a\rangle\) THEN, (read port, if equal to zero do 〈a〉)

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

The conditional specifiers for 6502 are:
\begin{tabular}{lll} 
CS & \begin{tabular}{c} 
test carry set \\
processor \\
status
\end{tabular} & \(\mathrm{C}=1\) in \\
OK & \begin{tabular}{l} 
byte less than zero
\end{tabular} & \(\mathrm{N}=1\) \\
\(0=\) & equal to zero & \(\mathrm{Z}=1\) \\
CS NOT test carry clear & \(\mathrm{C}=0\) \\
\(0<N O T\) & test positive & \(\mathrm{N}=0\) \\
\(0=\) NOT test not equal zero & \(\mathrm{Z}=0\)
\end{tabular}

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

\section*{ASSEMBLER DEFINITIONS HEX}

50 CONSTANT VS (test overflow set)

\section*{CONDITIONAL LOOPING}

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

6 L LOA, N STA,
(define loop counter in \(N\) )
BEGIN, PORT DEC,
(repeated action)
NDEC, \(0=\) UNTIL,
( N reaches zero)
First, the byte at address \(N\) is loaded with the value 6 . The beginning of the loop is marked (at 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．

\section*{CONDITIONAL EXECUTION}

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

PORT LDA，\(O=\mathbb{F}, \quad\)（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， \(0=\) 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：
\begin{tabular}{ll} 
PORT LDA， & （fetch one byte） \\
OK IF，\(N\) DEC， & （if \(D 7=1\), decrement \\
ELSE，N INC， & （ if \(D 7=0\), increment \\
THEN， & \(N\)（ continue ahead）
\end{tabular}

\section*{CONDITIONAL NESTING}

Conditionala may be nested，according to the conventions of structured pro－ gramming．That is，each conditional se－ quence begun（ \(\mathbb{F}, \mathrm{BEGIN}\) ，must be ter－ minated（THEN，UNTIL，before the next earlier conditional is terminated．An ELSE，must pair with the immediately preceding \(\operatorname{FF}\) ，
```

BEGIN, < code always executedD
C5 IF, <code if carry set)
ELSE, <code if carry clear>
THEN,
0= NOT UNTIL, ( loop till condition
flag is non-zero)
ccode that contimues onward>

```

Next is an error that the assembler security will reveal．
```

BEGIN, PORT LDA,
0= IF, BOT \mathbb{NC}
0= UNTIL, ENDIF,

```

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

\section*{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
PUSH add two bytes to the data stack．
PUT write two bytes to the data atack，over the present bottom of the stack．

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

CODE INVERT（a memory byte ）HEX BOT X）LDA，（fetch byte addreased by stack）
FF EOR，（complement accumu－ lator）
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 plecing it on the stack by：

CODE ONE（put 1 on the steck）
DEX，DEX，（make room on the data atack）
1 \＃LDA，BOT STA，（store low byte）
BOT 1＋STY，（ hi byte stored from Y since \(=\) 2ero）
NEXT JMP，END－CODE
A simpler version could use PUSH：
CODE ONE
1 \＃DA，PHA，（ push low byte to machine stack）
TYA，PUSH JMP，（ high byte to accumulator，pueh to data atack） END－CODE

The convention for PUSH and PUT is：
1．push the low byte onto the mechine 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 over－write the present bottom of the stack with the two bytes．Failure to push exactly one byte on the machine stack will dierupt execution upon usage：

\section*{FOOLING SECURITY}

Occasionally we wish to generate unstructured code．To accomplish this，we can control the sessembly 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 as－ sembler glossary．The－－－indicates as－ sembly time execution，and separate input stack values from the output stack values of the words execution．
\begin{tabular}{|c|c|c|c|c|c|}
\hline \multicolumn{6}{|l|}{\multirow[t]{2}{*}{BEGIN，\(=\Rightarrow\)
UNTIL，}} \\
\hline & & & & & \\
\hline \multirow[t]{4}{*}{IF， ELSE， THEN，} & \(\Rightarrow\) & & cc & －－－ & addrl 2 \\
\hline & ＝－＞ & addri 2 & & －－－ & addre 2 \\
\hline & \(\Rightarrow\) & addrl 2 & & －－－ & \\
\hline & or & addrE 2 & & －－－ & \\
\hline
\end{tabular}

The address values indicate the machine location of the corresponding ＇B＇EGIN，＇I＇F，or＇E＇LSE，．cc represents the condition code to select the processor status bit referenced．The digit 1 or 2 is teated for conditional pairing．

The general method of security control is to drop off the check digit and manipu－ late the addresses at assembly time．The security againat errors is less，but the pro－ grammer is usually paying intense atten－ tion 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）
（a）
CS IF，（leaves addrl and digit 2）
（b）
ROT（bring addrB to bottom）
JMP，（to addrB of BEGIN，）
ENDIF，（ complete false for－ ward brench from IF，）

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

\section*{ASSEMBLER GLOSSARY}
\＃Specify＇immediate＇addressing mode for the next op－code gener－ ated．
）Y Specify＇indirect indexed \(Y\)＇ad－ dressing mode for the next op－ code generated．
, \(X\) Specify 'Indexed \(X\) ' addreasing mode for the next op-code generted.
, \(Y\) Specify 'indexed \(Y\) ' addressing mode for the next op-code generated.
. A Specify eccumulator addressing mode for the next op-code generated.
\(0<\quad-\) ec (aseembling)
Specify that the immediately following conditional will branch based on the processor status bit being negative \((Z=1)\), i.e., leas than zero. The flag cc is left at assembly time; there is no run-time effect on the stack.
\(0=\)
-- ce (assembling)
Specify that the immediately following conditional will branch based on the processor status bit being equal to zero ( \(Z=1\) ). The flag \(c c\) is left at assembly time; there is no run-time effect on the stack.
;CODE Used to conclude a colon-definition in the form:
: <name>... ;CODE <assembly code> END-CODE Stop compilation and terminate a new defining word <name>. Set the CONTEXT vocabulary to ASSEMBLER, assembling to machine code the following nmenonics. An existing defining word must exist in name prior to ;CODE.

When <name> later executes in the form:
<name> <namex>
the definition <namex> will be created with its execution procedure given by the machine code following <name>. That is, when <namex> is executed, the address interpreter jumps to the code following ;CODE in <name>.

\section*{ASSEMBLER}

\section*{in FORTH}

Make ASSEMBLER the CONTEXT vocabulary. It will be searched first when the input stream in interpreted.

BEGIN, -..- addr 1 (assembling) 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, brench to BEGIN, will occur if the processor status bit given by ce is false; otherwise
execution continues thead.
At esembly time, BEGIN, leaves the dictionery pointer address addr and the value 1 for later testing of conditionery pairing by UNTI,
-.-. \(n\) (eseembiling)
Uned during code aseembly in the form:

BOT LDA, or BOT \(1+X\) ) STA,
Addresees the bottom of the data stack (containing the low byte) by selecting the , \(X\) mode and leaving \(n=0\), at assembly time. This value of \(n\) may be modified to another byte offeet into the data steck. Must be followed by a multi-mode op-code mnemonic.

CODE A defining word used in the form:
CODE <name>. . . . END-CODE
to create dictionary entry for <name> in the CURRENT vocabulary. Name's code field contains the address of its parameter field. When <name> is later executed, the machine code in this parameter field will execute. The CONTEXT vocabulary is made ASEMBLER, to make available the op-code mnemonics.

CPU \(\quad\) n..- (compiling aseembler)
An assembler defining word used to crate aseemblar mnemonics that have only one adoressing mode:

\section*{EA CPU NOP,}

CPU creates the work NOP, with its op-code EA as a parameter. When NOP, later executes, it assembles EA as a one byte opcode.
--. cc (aasembling)
Specify that the immediately following conditional will branch based on the processor carry is set ( \(C=1\) ). The flag cc is left at assembly time; there is no run-time effect on the stack.

ELSE,

Occurs within a code definition in the form:
ce IF, 〈true part) ELSE, <false part THEN, At run-time, if the condition code specified by ce is false, execution will skip to the machine code following ELSE,. At asembly time ELSE, ssembles a forward jump to just after THEN, and re-
solves a pending forward branch from IF. The values 2 are ueed for orror checking of conditional pairing.

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 exite the ASSEMBLER making CONTEXT the same as CURRENT. This word previously was named C ;

IF, \(\quad\) cc \(\cdots\) addr 2 (assembly time)
\[
\text { --. sddr } 2 \text { (assembly. }
\]
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, ( \(O<\) or \(O=\) or CS). If the specified processor status 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 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 IY 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 IY LDA,
loads the third byte ahead in the colon-definition being interpreted.

M/CPU nl \(n 2\)--- (compiling assembler) An assembler defining word used to create assembler mnemonics that have multiple address modes:

1C6E \(60 \mathrm{M} / \mathrm{CU} A D C\),
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.
--- addr
A variable used within the assembler, which holds a flag indicating the addressing mode of the op-code being generated.
--- addr (assembling)
Used in code definition in the form:
N1 - STA, or N2+ )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 \(\mathrm{N}-1\) thru \(\mathrm{N}+7\). 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, PUY, POP, POPTWO).
ccl --- ccl (assembly-time) 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'.

POP

POPTWO
\[
\text { nl } n 2 \text {--- addr (assembling) } \begin{gathered}
\text { (run-time) }
\end{gathered}
\]

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 \(\begin{array}{rlr}\text { (assembling) }\end{array}\) A constant which leaves (during assembly) the machine address of the return point which, at run-time, will add the accumulator (as high-byte) and the bottom machine stack byte (as low-byte) to the data stack.

PUT
\[
\begin{array}{lll} 
& --- \text { addr } & \text { (assembling) } \\
n 1 & ---n 2 \quad \text { (run-time) }
\end{array}
\]

A constant which leaves (during assembly) the machine address of the retum 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 ( \(n\) l).

RP) -- (assembly-time)
Used in a code definition in the form:
\[
R P) \text { LDA, or } R P) 3+S T A \text {, }
\]

Address the bottom byte of the return stack (containing the low byte) by selecting the,\(X\) mode and leaving \(n=\$ 101\). \(n\) may be modified to another byte offset. Before operating on the return stack the \(X\) register must be saved in XSAVE and TSX, be executed; before returning to NEXT, the \(X\) register must be restored.
--- \(n\) (assembling)
Identical to BOT, except that \(\mathrm{n}=2\). Addresses the low byte of the second 16 -bit data stack value (third byte on the data stack).

THEN,
addr 2 --- (assembly-time) Occurs in a code definition in the form:

> ce IF, <true part〉 ELSE, <false part THEN,

At run-time THEN, marks the conclusion of a conditional structure. Execution of either the true part or false part resumes following THEN,. When assembling addr and 2 are used to resolve the pending forward branch to THEN,.

UNTL,
... (run-time)
addr 1 cc --- (assembling)
Occurs in a CODE definition in the form:

> BEGIN, . . . ce 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.
    --- addr (assembling)
        Used in a code definition in the form:
UP LDA, or UP IY 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,
load the low byte of the sixth user variable, DP.
--- 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 ony byte in the definition's parameter field. i.e.,
\[
2 \text { F LDY, W )Y LDA, }
\]
fetches the first byte of the parameter field.

Specify 'indexed indirect \(X\) ' addressing mode for the next opcode 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.

```

CR 84
O (M/CPU, MULTI-MODE OP-CODES WFR-79MAR26)
: M/CPU <BUILDS C, , DOES>
DUP 1 + @ AO AND IF 1O MODE +1 THEN OVER
YPOO AND UPMODE UPMODE IF MEM CR LATEST ID.
3 ERROR THEN C@ MODE C@
INDEX + Ce + C, MODE Ce 7 AND IF MODE Ce
OF AND 7 < IF C, ELSE, THEN THEN MEM ;
1C6E 60 M/CPU ADC, LCGE 20 M/CPU AND, LC6E CO M/CPU CMP,
IC6E 40 M/GPU EOR, 1CGE AO M/CPU LDA, LCGE OO M/CPU ORA.
IC6E EO M/CPU SBC, LCGC 80 M/CPU STA, ODOD OL M/CPU ASL,
OCOC CI M/CPU DEC, OCOC EI M/CPU INC, ODOD 41 M/CPU LSR,
ODOD 21 M/CPU ROL, ODOD 6I M/CPU ROR, 0414 81 M/CPU STX,
0486 EO M/CPU CPX, 0486 CO M/CPU CPY, 1496 A2 M/CPU LDX,
OC8E AO M/CPU LDY, 048C 80 M/CPU STY, 0480 14 M/CPU JSR,
8480 40 M/CPU JMP, 0484 20 M/CPU BIT,
SCR 85
O ( ASSEMBLER CONDITIONALS
WPR-79MAR26 )
1 : BEGIN, HERE 1 ; IMMEDIATE
: UNTIL, ?EXEC >R 1 TPAIRS R> C. HERE 1+ - C, ; IMMEDIATE
IF. C, HERE 0 C, 2 ; IMMEDIATE
: ther, fexEC 2 ipaIRS here over ce
IF SWAP ! ELSE OVER l+ - SWAP C! THEN : IMMEDIATE
ELSB, 2 PPAIRS GERE 1+ 1 JMP.
SWAP HERE OVER 1+ - SWAP C! 2 ; IMMEDIATE
: NOT 20 + : (REVERSE ASSEMBLY TEST
90 COMSTANT CS (ASSEMBLE TEST POR CARRYSET )
DO CONSTANT O= (ASSEMBLER TEST POR EQUAL ZERO
10 COMSTANT O< (ASSEMBLE TEST POR LESS THAN ZERO)
90 COMSTAMT >= (ASSEMBLE IEST POR GREATER OR EQUAL ZERO)
(>0 IS ONLY CORRECT APTER SUB, OR CMP, )
SCR 86
O (USE OF ASSEMBLER
WPR-79APR28)
1: EMD-CODE (END OF CODE DEFINITION *)
CURREMT CONTEXT 1 ?EXEC ?CSP SMUDGE ; IMMEDIATE
TORTA DEFIMITIONS DECIMAL
: CODE ( CREATE WORD AT ASSEMBLY CODE LEYEL *)
\imathEXEC CREATE (COMPILE) ASSEMBLER
ASSEMBLER MEM !CSP ; IMMEDLATE
( LOCK ASSEMBLER IUTO SYSTEM )
ASSEMDLER CPA - ;CODR 8 + ! ( OVER-WRITE SMUDGE )
LATESI 12 +ORIGIN ( (TOP NFA)
HERE 28 +ORIGIN: (FENCE)
HERE 30 +ORIGIN 1 ( DP )
ASSEMALER 6 + 32 +ORIGIN ! (VOC-LINK)
HzRE FENCE !

```

\section*{APPLICATIONS}

\section*{A TECHNICAL TUTORIAL: TABLE LOOKUP EXAMPLES}

\author{
Henry Laxen \\ Laxen and Harris, Inc.
}

One of the problems with FORTH, as with every rich language, is that given an idea, there are many ways of expressing it. Some are more eloquent than others, but it takes practice and experience to create the poetry and avoid the mundane.

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
\begin{tabular}{|c|c|c|c|}
\hline 31 DAYS/MONTH & & & \\
\hline 28 'DAYS/MONTH & 2 & + & \\
\hline 31 'DAYS/MONTH & 4 & + & \\
\hline 30 DAYS/MONTH & 6 & + & \\
\hline 31 DAYS/MONTH & B & + & \\
\hline 30 'DAYS/MONTH & 10 & + & \\
\hline 31 'DAYS/MONTH & 12 & + & \\
\hline 31 DAYS/MONTH & 14 & + & \\
\hline 30 DAYS/MONTH & 16 & + & \\
\hline 31 DAYS/MONTH & 18 & + & \\
\hline 30 'DAYS/MONTH & 20 & + & \\
\hline 31 'DAYS/MONTH & 22 & & \\
\hline \begin{tabular}{l}
: DAYS/MONTH (I \\
1. 2* 'DAYS/MO
\end{tabular} & & & \\
\hline
\end{tabular}

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.
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 (NnNn-1 ... N1 n \(-\ldots\) )
ODO , LOOP ;
CREATE DAYS/MONTH
3130313031313031303128 3112 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?

\section*{CREATE TDAYS/MONTH}
\(31,28,31,30,31,30\), \(31,31,30,31,30,31\),
: DAY5/MONTH ( INDEX --- VALUE) 1- 2* 'DAYS/MONTH + ® ;

Now we are getting somewheren 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? Funny you should ask. There is a quality that can be abstracted from the definition of DAYS/MONTH, namely that of table lookup. Wouldn't it be nice if we didn't need to create that extra name 'DAYS/MONTH simply so we could access it later in our : definition. Well, that is where our friend CREATE DOES) comes in.

Instead of defining a particular instance of a TABLE, we will create a new Defining Word called TABLE, which acts as follows. It creates a new entry in the dictionary which when executed, uses the value that was placed on the stack as an index into itself and returns the contents of that location. It would be coded as follows:
: TABLE
CREATE \(\quad\) (…)
DOES
SWAP \(1-2^{*}+\) @ ; VALUE)

TABLE DAYS/MONTH
\[
\begin{array}{r}
31,28,31,30,31,30 \\
31,31,30,31,30,31
\end{array}
\]

Now we have truly generalized the problem and solve it in an elegant way. We have defined a new data type, called TABLE, which is capable of defining now words. Part of the definition of TABLE was specifying the run-time behavior of the word being defined. This is the code following the DOESD. We then use the , (comma) technique discovered above to initialize the table. Note that DAYS/MONTH is now just a special case of TABLE, and is in fact defined by the new defining word TABLE.

The above examples illustrate the immense diveraity 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 beat solution for themselves.

\section*{HZLP WANTED}

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

\section*{Contact: Patricia Jones}

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

\section*{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:

Now, let's look at another way of doing

\section*{THE GAME OF REVERSE}

> M. Burtan

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
\[
234516789
\]
and four numbers are reversed, the list will be
\[
543216789
\]
then if five numbers are reversed, the game is won.
\[
123456789
\]

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

\section*{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.

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

DIM \(n\).
A defining word used in the form \(n\) DIM \(x \times x x\)
Produces an \(n+1\) length word array named \(x \times x x\), 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 atring for an uppercase or lower-

228

```

O VARIARLE EEED (Seed \&or random number generator)

```
0 VARINDLE MOVEB (Number of feverees so far )
    CR ." Please depress any key:" (Fertilise the seed)
        KEY SEE ।
    RND ( Random number generator renge -- rnde )
        SED © 259 • \(3+32767\) AND DUP 8asm 132767 \%;
    DIM 1 Reserve an integer word arrey n -- 1
        <EUILDS 1 * 2 • ALLOT
        DOES> ;

229
The Gane of Reverse [Gen inatructionsl
                                    101281-MPB I
: Imstruct CR CR 10 spacrs ." The Game of REVERSE"
CR CR ." Hould you like instructions?" Y/M
IF ." The object of the geme is to arrange random iist"
    CR." of nine numbers into atcending nueerical order in"
    CR ." as fow moves as posibie by reversing a cubset of"
        \(C_{R}\)."the list. For exmmple, given the randem list." \(C R\)
        CR." \(\quad 5 \quad 2 \quad 4 \quad 8 \quad 7 \quad 3 \quad 9 \quad 1 \quad 6 \quad 1 . c R\)
        CR." reversing a subset, of would yiela the liet," \(C R\)
        CR."To quit the game, almply reveree \(0 .{ }^{\circ} \mathrm{CR} \mathrm{CR}_{\mathrm{R}}\)
    THIEX:
-->

```

ASCRANDLE ( M1x up the array valuen -- |
| Caleulate K |
Get Mmaylzl value I
Gat MRAy(K) value
| Etore mRAYIRI in MmayIII I
Etore NNAYi\i in mNAYiki )
+100P :
GETIN ( Cot mount te reveree -- n l
BEGIM CR ." Ruvarse hom manyt "
PAD }10\mathrm{ EXPECT PAD 4 -
DUP O< OVER }9>\mathrm{ ON DUP
IF CN " Oniy O through t is allowed. " THIm om
UNTIL CR : -->

```
232
The Gae of Reverse IARRAY operations, cent.j 1007e1-Mre,
AREVEREE ( Reverse aubeet of Matay n - 1
    DUP 2,141 (Loop limite are 1 te \(1 \mathrm{~m} / 2 \mathrm{iti}\) )
    DO DUP I - \(1+\) ( Calculate index \(n-I+1\) )
            DUP ae sidap \((\) Get MRRAY \((n-i+1)\) )
            ( Get RRRAYin-it+
( Get MRAy(I)
                    ( gtore NkRYiIj in NMAY(n-I*2)
                    ( 8tore RRRAY(n-I+1) in MRRAY(I) )
                            ( Check for ascending seq. -- fing )
    ACHECK
                    Ae \(=\) Not
                        LOOP :
-->
            1101 DO
                        ;
233
The Gene of Reverse (REVEREE dofinition
                                    101201-NPB )
    REVERSE (Play the gane)
        INSTRUCT MINIT
        BEGIN
            ASCRAMBLE 0 MOVES 1
                BEGIM
                    A. GETTN DUP \(0=\)
                    IF 1 E定
                        NREVEREE 1 MOVES +1 ACHECK
                    THIM
                UNTIL
            A. CR ." You made " MOVES © . ." reversals." CR
            CR ." Care to play again?" Y/N Oe
        UNTIL
    CR ." Thanke for playing REVERSE..." CR CR ; ; 8
E1g-FORTH Version 1.15
                            M. Burton
ok

\section*{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 anlyzing both the game itslef 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 bytes of memory. In fact, there are indeed extraneous "Cr's" which were included to get good hard copy, also.

This program was written in micromotion (c) FORTH-79 Version 1.2 to be rum on s 48K "Apple II.

Therefore, the following words are non-standard but included in the micromotion 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 anywhwere on the text window per Ex. 4 CV 10 CH moves the cursor to the 4 th (pun) row 10th column of screen.

SETINV, SETFLASH, and SETNORM eet 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 charcters already displayed.

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

Tony Lewis
100 Mariner Green Dr.
Corte Madera, CA 94925
(415) 924 -1481
(415) 924-4216 (late hours)

\footnotetext{
- Apple is a registered trademark of Apple Computer, Inc.
}

\section*{SCFW5 1}
: HOWTO: MONE ( 31 GAME-TOWY LEWIS) "" 31 GAFE EY TONY LEW 31' IS FLAYED WITH A DECK OF 24 CAKDS CONTAININING OMLY THE ACES THFU SIXES EACH OF TMO PLAYERS ALTERNATELY DRAWS CAKDS FFOM THE DECK. ONE CARD AT A TIME. A GLHNINE TOTAL IS KEPT OF THE CDNOINED"

SUM DF THE CARDS DRAWN. THE PLAYER WHO AKFIVES AT THE SUM OF 31 EXACTLY WING. IF NEITHER FLAYER CAN MANE SI EXACTLY THEN THE FLAYER WHO MUST GO OVEF II LOSES. THE GAME MAY APFEAF TOO EASY, BUT IT IS DECEPTIVE. WHEN TOF IF? Y YOU HAVE"
mun thfee games. try to beat the phogran FOR THE EIG RET' EY TYPING IN ' \(\mathrm{E}^{\prime}\) FGTMEK THAN ' \(Y\) ' DR 'N' HEN 'NEW GAME?" COMES UF. THE RIG EECY IS A TWO GAME SEFIES. YOU GO FIRST IN GAME I AND SECIND IN GAMI 2. YOU MAY EE SUAFRIEED:* CF CK CR ." HIT GNY KEY TO EEGIN KET DFDF MOME G CV

SCHES:
1 WOFDS OF WISDOM EI EY TONY LEWIS) 1 THE 'ANSWEF' FAGE IS NEXT. IT DOESM'T feGUIRE ANY SI ILL TO FIGUNE OUT WHAT THE CONSTANTS REALLY ARE, THEY AFE EMCODED SO TMAT YOU CAN ENTER OND COMFILE THE GAME WITHOUT DISCOVERING ITS PRINCIFLE. GEMEMPER THE PUPPOEE OF THIS PROJECT WAS TG GET YOU TO FIRST EXAMINE THE GOME GY P AYING IT THEN FIGU最E OUT MOW TO APGFUACM THE PROLEEM OF POOCRAMIING IT APFRUACH THE FROLLEM OF PROGRAMHIING IT AND FINALLY GO EACK AND COMPARE YOUR AND IS A LITTLE KNOWN CINCH BAF BET. IF AND IS A LITTLE KHON CINCH EAFE BET. IF YOU TAKE THE TIME TO ENTER IT ONTO YOU FOKTH DISC. YOU SHOLLD HAVE FUN BOTH GNALYZIMG IT AND THEN ENTERTAINING COR MUSTLINSJ FRIENDS AND FAMILY WITH IT. OF COUNSE WEN PLAYING AT A BAR YOU MUST USE A REAL DECK OF CARDS AS IT WOLLD PROBABLY TEND TO DISCOURAGE WMGERIMG IF YOU SHOULD BEING YOUR "WICRO" WITH YOU.'

\section*{SCR*S}
(ENCODED CONSTANTS 31 EY TONY LEMIS)
( NOIE: THESE CONSTANTS ARE USED GNLY
TO CONCEAL THE SOLUTION OF THE GANE.
NOT TO MAKE THE CODING MAFD TO FOLLDW!,
(i CONSTANT P: 1
a CONSTANT
CONSTANT +3
O CONSTANT 1.4

HE!
CODECONS
CDEF ABEC: - \(41 C E-\) ki! \begin{tabular}{l} 
DCFE BACE - \(46 C 7-\quad .12\) \\
\(C E E D\) \\
\hline
\end{tabular} EC DE CA DE - IFDE - K4: DECIMAR

\section*{SCFw 4}

SETUF AND UTILITY WORDS 3i-TONY LEWIS: CfEATE DEC
6.4.4.4.4.4.4.0

VABIABLE CAKDSUM VANIAELE BANESHON
```

NEWGANE I FIRST. NEW DECK
7 1 DO, 2\# DECK. \& SWAP: LOON
G CARDSUM, MOME \& CV,

```
SHOWDEC.SUM CR CR
    " THE DECR MOW CONTAINS -
    71 DO 1 ( NOT 3!! 1 2t
        DECK + CO D DUCK a dO) DUP
                IF 1+ 1 CF DO 3 . LOOP
                    ELSE DROP
                        THEN
            COOP 12 CA
                            ." the rumwing total 15 " cardgun a. :
: Badplar
( FLAG RAD flay 0 CR CR
." BAD TYPE-IN" SHOMDECKEUM:

SCKms5
( UTILITY WORDS CONT. 31 BY TONY LEWIS
UFDATEDECK ELM 1 TO 6 MOW ON STACK, DUF 2\% DUP DECk + Ca DUP (ArAY LEFT?) If 1- ( UPDATE DECk.) Smaf DECk 4 C. CARDSUM C2 - CAFDSUM L' (NEW SUM: 2 ( CAFD-IN-DECI. FLAG)
EISK LHOF LFOF DFOF O
THEN:
S. Tr 'to

Motlia.JNE WORDS
PLATERMOVE CR CR
." Tyfe in lafij 1-6"ker ck
\(4 E\) - FROM ASCII, DLF DUF DUF
' CHFCK VALID ENTFY,
7 IF
F i:
HOMTEDECRSUM SWAF
MONE :" YOUK CAED HAS A ". DUF
Lo 'IS CAFEL IN THE DECR. T,
it EfiOF : FLAE EMD PLAY) O
CR CH." CARD NOT IN DECK. SHOWLEC*SUM THEN
ELSE DKOF DFOF GADHLAY
THEN
ELSE DROF DROF DRUF EADPL.AY
THEN:

SCW制 7
(MAINLINE WOKUS CDNT. IS EY TONY LEWIS
- mriakll CR CR

IF , CHECF 15 T fisiy SulTiH
: RNatDiM it DUF IPDATEDECI.SUM DROF
ELSE 11 DUF CaFDSUH C. *
LFDATEOCRSAM
diateprcrsin
LV: T FLAG ON VALID CHOICE
DF()W 1 Eur IPDATELECI.SUM
If
E:SE UNOF : UFDATEDECR SUM
Tilin : FLAG=Catio. Su No bfict Theis
there ." Hy Miny is ". Smumbert ind
--
siram

renturamer


UWIIL SHOWDF C: StM
" F Flayfo male last play. :
: VLewin in ra selin.
""Unt kirs" yous win. - Sernotin

- YOLLOSE Cf CH beting
-" vin luse. EETIEf liKA reat time. SEINOFM:
sCKins.
- MAIMLIME WOFDS LUNT. 31 er TONY LEWISI

MOFTHAL 31 MONE 7 CV
- "DO TLU WANT FIGsit flat" Trfe O OK N.

CF HEY \(76=1 \mathrm{~N}\)

THEN O SHAN , SET UF LOOP.
DEGIN
\&F , IRUE PLAG SET ON MY( AFO:
31 CAEDSM Li
If YOUWIN it, GET LOCir EAII,
ELSE : 1 CAROSUM CO \(=\)

IHEN
IHEN
GI SE ( RE HENT FROM YOURCAFE,
21 Cand
if YOULUSE \(1+\)
ELSt El i MAtrsum \(\mathrm{Ca}=\)
IF YOUNIN 1+ ELSE ? MrCAES THEN , NOO IST HOVE,
TrIEN
TimeN
WIIL 0 ( LưF back in maldsil:
scrubco
MAINLINE WORDE CONI. 31 BY IONY L.EWIS
MYBIGBETI CR CR
\(k 1\) DUP CARDSUM CA \(+k 2-k 1\) MOD -
LUF UPDATEDECKOUM O*
IF DROP KJ DUP UPDATEDECKSUM DROP THEN ." MY PLAY 15 " . SHOWDECKPLH 3

MYEIGRET2 CR CR
IF DUF UPDATEDECKBUM DROP
ELSE KI CARDSUM CD K2 - 11 MOD - DUP UPDATEDECK:EUM OM

IF DROP K 4 DUP UPDATEDECKSUM DROP HEN
THEN ." MY PLAY IS". SHOWDECNSUM 3 :
--
SCRW6 1
MAINLINE WORUS CONT. SI HY T(HNY LEWIS) BIGEETI
VOURCARD
ET.GIN
IF (TRUE FHOM MYEISELET
\(\therefore\) CARDSUM L. .
IF YOUWIN 1 ( ETET LOOF EXII)
ELSE Z1 CARDELH L゙O 」
if YOULOSE 1
ELSE VOURCAFD O
THE.N
IHIN
rife ( REIUFN FKGA P(IURLAREL)
1 CARDSLMM LI
F YOtLUGF
Li: VOUMIN 1
ELSE MYELOLET
ELSE MYEIGEET1 \%
That
Hirn
ifirn
imen
UNOIL:

MAINLITEE WOKDS LIJNT. if BV TUHIY LEWIS)
O!GEEI:
F MrEIGEET.
EEGIM
It \(=1\) LAKDEstm i. \(\omega\). \(x\)
IF YOLLUSE 1
tLESE VUJUKEAKD "
THEN
TLSE 31 LARDSUM i. 3
if roth OSt 1
 THEN
THEN
JJNTIL:
ilkMOS
MAIPILINE WUKDS CINTI. II EY ? UNY LEEWIS:
HIGREI 5 HDME
WEt COME 10 'EIE EET'. THE FINAL PWASE OF THE 31 GAPIE. I WGO GOWH WIt BE FIAYEO." GR ." YOU WILI GU FIRSI IN GAME GNE ANNL Will hod first in bamt iwit Gowit tht..

Ellizer Tit
IN IR
HII ANY tEY AND I WII REEGIN BAME 2.
in tey thiof Newtimimb
SEFPIASH ." HIG KEI IGMN a THE FINATE"
،F LH SET TAMRM
EHIGET2
Le

 1. F

WHO WINS WITH A ISY LAFET TIF ONE UH IWU.
IT \(S\) A TOUGH CUMBINATCHEIAL HRDEAEM"
1, BFT FINAI. FXIT IN MAINSI):

SCRW64
( PLAY THE GAME OF 31 DY TONY LEMIS

EEGIN CR CR
" MEW GAMEs "
"TYPE Y OR N OR PIBIC EET)."
CR CR KEY DUP 70 - ( CHICK PON N)
"FALBE LEAVES 74 ON BTACK FOR "UNTIL")
IF NEWEANE to (D)
IF SANESHIN Co 2 ?
IF HIOBET
ELEE HOHE 0 " YOU MAVE WON *
Garteswow Cs Dup. "" gavt" 1 m
IF.""."
THEN CR
" YOU MUBT WIN 3 gante TO * " play 'die ert'."
THEN
El.BE NORMAL 31
THEN
THEN
SIGAME HOWTOZI CODECON MAINS:

OF:
CODECONSA MAINBI
NEW GAME? TYPE Y OR N ON DSBI LETJ.
DO YOU WANT FIRAT PLAYT TYPE \(V\) O. \(N\).

MY PLAY 182
THE DECA. NOM CONTAIME
1111
\(\therefore 22\)
\(\begin{array}{ll}4 & 4 \\ 5 & 5 \\ 5\end{array}\)
60 THE RUNNINS TOTAL 192
TYPE IN CARD \(1-6\)
YOUR CAFED WAS A 3
```

THE DECT NOW EONTAINS
1 1 1
3. 5
O% THE RLNNING ralth 15s
my plar is 5
THE DECK NOW CUNTAINS
!!!1
2
4}44
O}\mathrm{ O SO HHE FLONNINGS TOTAL IS 15

```
    TVHE \&N EHFD \(1 \cdots\)
    YNT CARD WHS HC
    HE LEET THIW CONIAINS
    1111
    \(\therefore 7 \%\)
44
4
    535
    ris THF RUNWING TUTAL IS 10
    my flay is 1
    THE DECK. MMW CONIAINS
    \(\begin{array}{ll}1 & 1 \\ 2 & 1 \\ 2\end{array}\)
    222
    444
    \(\begin{array}{lll}4 & 4 & 4 \\ 5 & 5 & 5\end{array}\)
    65 THE RUNNINE TOTAL IS 17
    TYFE IN CARD \(1-6\)
YOUF CARD MAS A 3
THE DECK NOW CONTAINS
\(\begin{array}{lll}1 & 1 & 1 \\ 2 & 2 & 2\end{array}\)
\begin{tabular}{l}
1 \\
2 \\
2 \\
\hdashline 3
\end{tabular}
4444
55

The gunnaing rutal is 20

HY PLAY 184
THE DECK NOW CONTAINS
\(\begin{array}{lll}1 & 1 & 1 \\ \vdots & 2 & 2\end{array}\)
\(\begin{array}{ll}2 \\ 3 & 2 \\ 3\end{array}\)
\(\begin{array}{lll}4 & 4 & 4 \\ 5 & 5 & 5\end{array}\)
\(\begin{array}{ll}555 \\ 6 & 6\end{array}\)
TYOE IN CAKD 1-6
YOUF CAKL WAS A 5
THE DELP NCH CUNTHIN:
111
222
\(\therefore 3\)
\(45^{4}\)
Ois THE RIMNITAG TOTAL \(1: 29\)
MY Pl.hy \(15:\)
1HE DELK HOW CONTAINS
111
72
\(\therefore 3\)
\(\begin{array}{ll}4 & 3 \\ 4 & 4 \\ 5 & 5\end{array}\)
St THE RUNNINE IUTAL I5 \(\therefore\) :
YOU LUSE. BEITER LUK, NEXT IIME.
NEW GAWE, TYPE Y OK N IJR HCHIG BrTt.
YOU HAVE WON í GAMES.


U.

\section*{FORTHCLASSES}

LAXEN AND HARRIS, INC.
24301 Southland Drive
Hayward, CA 94545
(415) 887-2894

Introductory clasies
Procese control
Applications programming
Systems level programming

GREG STEVENSON
Anahe im, CA
(714) 523-4202

Introductory classes

KNOWARE INSTITUTE OF TECH-
NOLOGY
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

\section*{SIMALATED TEKTRCNCS} 4010 GRAPHICS WITH FORTH
by Timothy Huang Portland, OR 97211

In this article, 1 am going to tell a true story. For those people wh think FORTH is a religion, they might just consider this to be my testimony.

Last November, 1 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 Z-80A 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 \(512 \times 256\) bit mapped graphics. It also includes a \(\mathbf{9 "}^{\prime \prime}\) 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 examle: "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. \(512 \times\) 256 bit mepped graphic is the best that can be expected under the price allowance. There are quite a few wall 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. Anxioty mounts up quickly.

I have a friend who's an excellent 8080/Z80 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
```

screen t }1
O Video coatrols iog Columead M8764 TVM 2%18%/8\&;
FORTH DEFIMITIONS DECIKAL
cotexy ( y y -.- ;
0 Max 23 HiN 33 - guat
4 mAX 79 MIM 3* *
If EMIT EMI% EMIT

```


```

        RZ-C 12 EMIT. . BELE 7 EMET.
        CLREOS 23 EMIT . CFREOL 22 EMIT.
        CLAEINE 21 EME%.
    Scremn t1:
        , iraphic fackate = : mDK 1%/3%180;
    O VARIABLE \& O VAR:ADLE % valiABLEL ovagRaMLE %
O VARIABLE EI O VARIAgLE Y:
ESC :% EMIT , IT 12 EMIT, 6s 29 EM1% i vectuz, ;

```

```

        EM 23 [MIF ( eleat vicee memerf)'.
        UHITE &SC D7 EHI%. : SLAEK E35 i27 EXI%
        10
    14
    ```


```

        XGEN (4x, waty m)
            32. EMIT 4!. ENIT
    ```

```

        &RE-0ut < u,N-\
            vEcTON YGEA vECFOR EGEN.
        PAGE ( enter dipha isot vectgr)
        EgC It.
    sereer l

```








```

    s
    i2
Screen - 12
* (T)

```

(wo at least knew that the graphic part simulates Tektronics 4010), he spent a whole week just trying to draw one mere equare along the edges of the CRT. SeemIngly it would be an easy job, but even ao it never came near to what he would have liked. Later on, I apent 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 figFORTH 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 1 am steadfast in my belief in FORTH.

Screen 10 and 11 sets up the variables and the Columbia Mx964 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 ss 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.


\section*{A VIDEO VERSION OF MASTER MIND}

David Butler
Dorado Systems
The writing of this program served as my introduction to FORTH. Using the figFORTH 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 Mester Mind, therefore, all credit for the game itself goes to the Invicta Game Company.

The program containa 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 displeyed. 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
    (
\(-->\) David A. Butler
    33300 Mission Blvo
        Aft 126
        Union City. CA. 94587
        (415) 487-6039
    **** A note about strie: If there is anr, it is an accident.
    This was mr first application in Forth, so it mar lack
    some elerarice.
**** Requirements: A videc dismlay \(80 \times 24\) characters,
                                    cursor addressing and clear screen
                                    functions.
This is an implementation of Master Mind by Invicta.
The same is verr popular because it is easr to learn and a
challense to plar. There is a bit of luck to it, but it is
mainly an exercise in losical deduction. A "secret" code is
senerated, and it 1 s "cracked" by analrzins set of clues.
            Thase familiar with the orisinal board same will have nig
difficultr adjustins to the computer version. To newcomers,
folluw the directions carefully and rou will have it in no
time. The Forth version is functionalir identical to the
board version. It is written in fis-Forth, and has been run
successfulliv on 6502, 80s0, 280, and 68000 processors. It
is a sood demonstration prostam as well as an enjorable same.
SCR 20
(Master Mind set up some variables DAB-17nov30)
: TASK : ( FORGETTABLE MARKER )
0 VARIABLE COLORS 28 allot COLORS 30 BLANKS
O VARIABLE SCODE 2 allot 0 VARIAELE GUESS 2 allot
O VARIABLE SECRET 2 ALLOT
O VARIABLE BLACKER O VARIABLE WHITER
6 VARIABLE ECLOR'
3 VARIABLE CUR. ROW 23 VARIABLE CUR.COL
1 VARIABLE XLOC 1 VARIABLE YLOC 0 VARIABLE DONE
-->
* 21
(Master Mind set up - cont. DAB-17novBO)
: C.CONSTANT ." YELLOWFED BLACK GFEEN WHITE BLUE "
O VARIABLE COLOR.KEY 6 ALLOT ("colors" table)
(Use the sum of the ASCII code of the first 3 letters )
(i.e. BLUE \(=" B "+" L "+U N=66+76+85=227\)

234 COLOR.KEY C! 219 COLOR.KEY \(1+\mathrm{C}\) !
207 COLOR.KEY \(2+C!222\) COLOR.KEY \(3+c\) !
232 COLOR.KEY \(4+c!227\) COLOR.KEY \(5+c\) !
96 COLOR. KEY \(6+C\) !
O VARIABLE MATTEMPTS ( used to keep score)
    -->
```

* 19
( Master Mind
-notes-
DAB-17nov80 )
-->
* 

```
The inne is verr populartion
15
\(\cdots\)
-->


```

SCR
22
Master Mind prompt and randomize DAB-i7nov80)
( These definitions set the random values for the seme )
N NEWCOUNT ( [COLORW + !] ) DUP MCOLORS < <
IF 1+ ELSE DROP 1 THEN :
RAND 1 BEOIN NEWCOUNT ?TERMINAL UNTIL KEY DROP :
: ASK.FOR.RANOOM ." To randomaze, tap gmace bar a times."
4O DO RAND I SCODE + C! LOOP CR ;
: ASK.FOR.LEVEL
CR." Level 1 OR 2 ? " KEY DUP EMIT KEY EMIT
50 = IF 7 MCOLORS ! ELSE 6 \#COLLTRS ! THEN CR :
-->
|
Master Mind translate color to numeric
DAB-1;novg()
COLOR.FIND ( [COLOR*] ----L] TYPES COLOR FROM %)
1-6-C.CONSTANT 3 + + 6 TYPE ;
TRANSLATE. CODE
(converts color from SCODE to COLOR.KEY)
( numeric value in array "SECRET" )
4 O DO SCODE I + LE 1 - COLOR.KEY + CU SECRET
I + E! LOOF;
:J R: R: R [COMPILE] R SWAF \PR SWAF \R SWAF \R ;

# 24

    Master Mirid cursor motion nAE-17noveO,
    ( Gif course, CRT dependent. Here is Heath:
: CURGOR ([Y] [X]---[] ARSOLIME CURGOR FOSIT]INN)
\Xi1 + SWAF 31 + S% 27 EMIT FMIT EMI' EMIT :
: GLEAR ( ELEAF CRT SGEEN ) 27 EMI: E.9 EMIT ;
: HOME ( FUIT OIRGOIR AT HOME FUSIIIIN, O O IURCINF :
*** end at OF'i dependert words ***)
BHOW.IOLORE ( IISFLAY GOLGF EHOICES;
7 1 [IO I 2 + 59 CURSOR I GOLOR.FINII LOHF
\#LGLOKS E 7 = IF % 5% CLIRSUF." <ELAANK`" ELSE THEN
12 5s EUR"atR ." TAB between colors,"
13 S8 GLIFSNR ." RETURN to get -lues." %
--
< <
Master Mirid toard larnut mask [MA-17rmvi%O,
HAF ." :" ; : DASH ." -" ; ( FOAFFL EYMELLS ,
TITLE 21 SFAIES
." ==== M ASTER MI N | ==m=" ;
[IASHER 2 21 LOR:OOR EAR 32 " DO ." ~" LIUF FAK IR :
CLINE DUF 21 CURSOF BAF 54 CURSGR GAR ;

```


```

DISPLAY. BCIARD
CLEAR TITLE DASHER HIDDEN 24 }\because\textrm{DO}\mathrm{ I GBLOCK 2 +LOOF
SHOW.COLORS ;
-->

```
abling the player to koep changing his guea until he la satisfied that it is consistent with the clues he has thus far recelved. A correct guess is the result of the player'a logical deduction (or very good luck) based on his previous clues. The directione on acreen 31 explain the meaning of the two types of clues.

When the player signals he is ready, the program compares the player's guess to the secret code which was stored away oarlier. 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 take 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.

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

SCR

# 2t

( Master Mind cursor trackins definitions DAB-i7nov80)
: x XLOC e : : Y YLOC :
< XRIMMF X 52=
IF 23 LUP CUR.COL ! XLOC !
ELSE 1 XLOC +! X CUR.COL e 8 + x
IF X CUR.COL ' THEN
THEN ;
: IINEIJMPX X 23 = IF 5% XLOLC ' ELSE -1 XLOC +! THEN ;
TAB CUR.COL \& 47=
IF 23 CUR.COL'
ELSE \& CUR.COL +!
THEN CUR.COL XLOL ! DKOP Y X CURSOR ; --s
27
Master Mind character collection/editans DAB-17noveg,
BACKSFACE x CUR.COL e =
IF DROP
ELSE UNGUMPX Y }x\mathrm{ CURSOR SPACE }Y\times\mathrm{ CURSOR DROP
32 COLORS X + 23 - C'
THEN ;
: PROCESE ([CHAF] -- [] PROLESSES CHAR, MANAGES CURSUR )
DUP EMIT COLOFS X + 23-E: XBUMP Y x CURSGIR ;
GET.CHAF KEY DUF 127=
IF BACKSPACE ELSE LUF }\overline{y}
IF TAB ELSE LUNF 13=
IF I DIONE, DROF
ELSE fROCESS THEN THEN THEN ;
-->
SCR * 28

```
```

            Master Mind
    ```
            Master Mind
                    suess / row sectiop
                    suess / row sectiop
                                    LAK-17riov80,
                                    LAK-17riov80,
INITIAL 2 - 3 + LMJP YLUC ! CUR.ROW ! 23 23 XLOK
INITIAL 2 - 3 + LMJP YLUC ! CUR.ROW ! 23 23 XLOK
            ! CUR.COL ! Y X CURSOF
            ! CUR.COL ! Y X CURSOF
            30 O DO 32 1 COLORS + C! LUOP ;
            30 O DO 32 1 COLORS + C! LUOP ;
GET.COLORS INITIAL O DONE ! BEGIN LIET.IMAF IMNE E UNTIL ;
GET.COLORS INITIAL O DONE ! BEGIN LIET.IMAF IMNE E UNTIL ;
PARSE.LIUESS 4 O DO I 8 COLUFS + CP
PARSE.LIUESS 4 O DO I 8 COLUFS + CP
                    I 8 * COLORS 1+ + Ce
                    I 8 * COLORS 1+ + Ce
                    1 8 COLOF: 2 + + Ce
                    1 8 COLOF: 2 + + Ce
                                    + + I GUESS + E! LOOW ;
                                    + + I GUESS + E! LOOW ;
-->
-->
    29
    29
        Master Mind Clue seneration LAB-17nov&0,
        Master Mind Clue seneration LAB-17nov&0,
    CLUE.CHECK
    CLUE.CHECK
    O BLACKER '... O WHITEF ! ( INITIALIZE COUNTS ,
    O BLACKER '... O WHITEF ! ( INITIALIZE COUNTS ,
    4 0 LIM
    4 0 LIM
        SECRET I + CE GUESS 1 + CE = ( CHECK FGIF IITRECT HIT )
        SECRET I + CE GUESS 1 + CE = ( CHECK FGIF IITRECT HIT )
        IF 1 BLACKEF + ! O I LUUESS + C!
        IF 1 BLACKEF + ! O I LUUESS + C!
    THEN LDOP
    THEN LDOP
    40 DG GLIESS I +IE 0 > IF i IF NOM HIT,
    40 DG GLIESS I +IE 0 > IF i IF NOM HIT,
            40 DO
            40 DO
            GLESS I + CE BECRET J + CE = ( GHELK FOR WHIIE )
            GLESS I + CE BECRET J + CE = ( GHELK FOR WHIIE )
            IF I WHITEH +! I { GUESS + !? LEAVE
            IF I WHITEH +! I { GUESS + !? LEAVE
            THEN
            THEN
            LDMP THEN
            LDMP THEN
        LnOF
        LnOF
-->
```

-->

```


\section*{TRANSFER OF FORTH SCREENS BY MODEM}

\author{
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 FIG 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 19,200 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 SNPUT. 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 containe 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 line. 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 ust in case the receiver sends one or more OK's back to the sender during transfers, 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 tranamitter wait for this carriage return. This is the only handshaking needed. Each line's text is sent preceded with the letter \(P\) and a space, and followed by a carriage return. WAIT causes the tranamitter 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 end recorded on disk will correspond exactly to the screen numbers sent.

If that is inconvenient, a variable containing an offect 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 thundred data entries batween my machines with no trouble.
```

bl.f \# C.8
| Surlal Screen transter -- sending GTG 7-0Z-BI
HEX
: suturFul = 2S22 C! :
: COLITFUT 2 2322 C':
: SINFUT 1 2321 E!;
: CINPITT 2 2S21 C: :
SGUT SOUTPUT CINPUT SIN COUTPUT SINPUT
: OK ; : WAIT SIN QUERY ;
: SEND.SCREEN (SCR年 --> nothing left)
GOUT DUF . . " COUTPUT LIST SOUTFUT " CR WAIT
IG O DO 1 SOUT . ."P " I OVER .LINE CR
WAIT CINPUT ?TERMINAL IF LEAVE THEN LOOF :
: SEND ( FIRSl SCF* / LAST SCR\# --* nothing lett v
1+ SWAF DO I SEND.SCREEN TTERMINAL. IF LEAVE THEN LOOF
SOUT CK WAIT SOUT ."FINISHED " CF COUTFUT ;
DECIMAL :S
SCR * BO
(CONSOLE/SERIAL I,O )
FORTH DEFINITIONS HEA
: UNLINK FDFO SS! FDIE 3B!;
: SOLTFUT CZOOO Jó ! ;
: LIUTPUT FDFO 36 : :
: SINFUT C200 38: :
: CINFUT FDIE 3B:
EWITOF DEFINITIDNS
: FINISHED CINPUT COUTPUT FLUSH;
: F COUTFUT F SOUTPUT :
: RECEIVE COUTFUT SINPUT;
FORTH DEFINITIONS EDITOR
DECIMAL
S

```

\section*{HEP WANTED}

Part-time - Now Yark-Now Jermoy Area
Assist internationally known sound artist, Max Neuhaus, develop additional software for micro computer 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, NV 10010

Independent FORTH progremmers to implement Marx FORTH for TRS-80, 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 Califomia to do it.

Contact:
Marc Perkel
Perkel Software Systerns
1636 N. Shermen
Springfield, MO 65803
(417) 862-9830

\title{
PROOUCTS REVIEW
}

\author{
SORCERER-FORTH \\ by Quality Software
}

For about a year, I have been uaing 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 \(8080 \mathrm{fig}-F O R T H\). 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 en-courages--users to market application programs incorporating Sorcerer FORTH. They do ask that written permission be obtained frist, 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.

\section*{C. Kevin McCabe}

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

\section*{A COMPARISON OF TRANSFORTH WITH FORTH \\ Insoft \\ Medford, OR}

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

\section*{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.

\section*{Transcendental functions}

The floating-point format mentioned above makes TranafORTH natural language for tranacendental functions. Functions included in the system which are not found in most veraions of FORTH include: sine, cosine, tangent, arctangent, natural logarithm, exponential, square root, and powers.

\section*{Dats 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.

\section*{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.

\section*{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.

\section*{Grahics}

Two graphics utilities along with a couple of graphics demo prograns 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 systern 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.

\section*{Vocabulary}

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

\section*{Compilation and apeed}

All entries in TransFORTH are compiled directly into 6502 machine language for greater speed. No address interpreter is used. Even immediate keyboard entries are compiled before being executed. This means that ro:stines can be tested at the keyboard for speed before being added as colon definitions.

TransFORTH is fast. It is not as fast an integer versions of FORTH, because it handes 5 bytes with every stack manipulation instead of two. TransFORTH programs will run faster than similar Applesoft 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.

\section*{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 CONVERS group of languages, as it compiles to assembly language rather than threaded cade.
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 FOR TH 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.--ed.

\section*{NEW PRODUCTS}

\section*{FLEX-FORTH}

Complete compiler/interpreter, assembler, editor, operating system for:
\begin{tabular}{ll} 
APPLE II computers & \(\$ 25.00\) \\
KIM computers & \(\$ 21.00\)
\end{tabular}

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 16 K starting at 1000 HEX and often as little as 12 K , including the FLEXFORTH 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, \(\mathbb{F} . . . E N D I F\), 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 casette 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 assambly code.

Order from: GEOTEC, 1920 N. W. Milford Way, Seattle, WA 98177. Be aure to specify machine.

\author{
MARX FORTH V1.l \\ Perkel Software Systems \\ 1636 N. Sherman \\ Springfield, MO 65803
}
(417) 862-9830

Enhanced Z80 fig-FORTH implemented for Northstar System enhancements include linix 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.

\section*{Prices \$75.00}

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

\section*{TWO NEW PROOUCTS FROM LAXEN AND HARRIS, ENC.}

Laxen and Harris, Inc. 24301 Southland Drive
Hayward, CA 94545
(415) 887-2894
1. Working FORTH

Release 2.1
"Starting FORTH" compatible FORTH software for a 8080 or \(Z 80\) 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:

Compiters
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 \(\mathbf{Z 8 0}\) computer system with at least 32 K 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 14 K bytes of memory which includes 4 K 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: \$33.00 - phas \$2.00 - Poetage and Hendling
\(C P / M\) is a registered trademark of Digital Research, Inc.

\section*{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 thet 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 figFORTH 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 \(8^{\prime \prime}\) single density diskettes in IBM 3740 format. The price is \(\$ 33.00\) if ordered together with the Working FORTH Disk. Plesse add \(\$ 2.00\) for shipping and handling, and allow at least 3 weeks for delivery.

Note: Buy both for \(\$ 55.00\) plus \(\$ 2.00\) poetage and hendling.

\section*{POL YMORPHIC FORTH}

Abstract Systems, etc.
1686 West Main Road
Portsmouth, RI 02871
(401) 683-0845

Ralph E. Kenyon, Jr.
Product Description: FORTH (PolyMorphic fig-FORTH 1.1.0). 8080 figFORTH 1.1 without asmb. 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, 8810 needs only 16K. Documentation on FORTH not included.

Manual: documentation covers particular implementation details for figFORTH to interface to the PolyMorphic Systems Microcomputer. Sorted VLIST included.

Implementation document available separately. Separate document avaiiable for cost of postage. Product data available on PolyMorphic SSSD 5" diskette format. 4 copies sold to date. Price: \(\$ 40.00\), 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).

\author{
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 H 89 with 32 K 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 loops), an-line tailoring of FORTH facilities (e.g., number of disk buffers), ability to hook to separately assembled routines, and use of Heath DBUG.

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

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 \(5-1 / 4^{\prime \prime}\) disk for \(\$ 25\) including documentation. Documentation is available for \$4.00. (Conn. residents please add sales tax).

\section*{NEW PRODUCTS FROM TNER ACCESS CORPORATION}
1. Fig-FORTH compiler/interpreter for PDP-11 for RT11, RSX1IM or standalone 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 \(8^{\prime \prime}\) single density diskette only.

Reference Manual for PDP-11 figFORTH above. Price: \(\$ \mathbf{2 0 . 0 0}\)
2. Fig-FORTH compiler/interpreter for \(C P / M\) or CROMEMCO CDOS 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 \(1 / 4^{\prime \prime}\) diskettes in 128 byte, 18 sector/track format and \(8^{\prime \prime}\) diskettes in 128 byte, 26 sector/track (IBM) format.

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

Released on one \(8^{\prime \prime}\) diskette with source in 8080 assembler.

Released on two 5-1/4" diskettes with source in Z 80 assembler.

Released on one \(8^{\prime \prime}\) diskette with source in Z 80 assembler.

Manual for CP/M (or Cromemco) figFORTH above. Price: \(\mathbf{\$ 2 0 . 0 0}\)
3. METAFORTH \({ }^{T M}\) Cross-compiler for CP/M or Cromemco CDOS to produce 79-Standard FORTH on a target machine. The target can include an application without dictionary heads and link words. It is available on single density diskettes with 128 byte 26 sector/track format. Target compiles may be readily produced for any of the following machines:

CROMEMCO-all models
TRS80 Mode! II under CP/M
Northstar Horizon
Prolog 280
Released on two 5-1/4" diskettes or on one \(8^{\prime \prime}\) diskette.

Price: \(\$ 450.00\)
4. Complete Zilog (AMD) Z8002 development system that can be run under CP/M or Cromemco CQQS. System includes a METAFORTH \({ }^{M}\) cross compiler which produces a 28002 79Standard FORTH compiler/interpreter for the Zilog Z8000 Development Module. Package includes a 28002 assembler, a Tektronix download program and a number of utilities.

Released on two 5-1/4" diskettes or on
one \(8^{\prime \prime}\) diskette.
Price: \(\$ 1,450.00\)
5. Zilog Z8002 Development Module figFORTH ROM set. Contains 79-Standard FORTH with Z8002 assembler and editor in 4 (2716) PROMS. Price: \(\$ 280.00\)

CODE9
Arthur M. Gorski
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 requirenents: 4.75 K bytes free RAM above FORTH. COOE9 is distributed as a commented source listing and manual. Price: \$20.00

> PET-FORTH
> by
> Datatronic AB
> Box 42094
> S-126 12 Stockholm
> Sweden
> (0)-8-744 5920
> Peter Bengtson

Product Description: Extended figFORTH 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, 8 very complete string package, IEEE control words, integer trig functions.

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

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

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

Manual Description: 322 pages, including all source code.

Complete introduction to FORTH. Special chapters cover the asembler, 〈BUILDS and DOESD, 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.

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

Prices \(\$ 290.00\) Includes diskette, ROM, manual, shipping end 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 Petalect Electronic Services Ltd, 33/35 Portugal Road, Woking Surrey. You may also order directly from us.

\section*{Diskette of FORTH Application Madules from \\ Timin Engineering Company 9575 Genessee 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, <BUILDS...DOES> 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 epplication modules is \(\$ 75.00\) (if other than \(8^{\text {" }}\) standard diack, 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.

\section*{AUDIO TAPES OF \\ 1960 FORM CONFERENCE AND 1980 FIG CONVENTION}
1. FORTH-79 Discussion, 200 min . Prices \(\$ 35.00\)
2. Purpose of FIG, 37 min . Price: \(\$ 10.00\)
3. Charles Moore, 63 min . Price: \(\$ 15.00\)
4. FORTH, Alan Taylor, 47 min . Price: \(\$ 15.00\)

\section*{Complote set \(\$ 65.00\)}
edu-FORTH
1442-A Walnut Street, 332
Berkeley, CA 94709

The following vendors have versions of FORTH availeble or are conoultenta. (FIG makes no judgment on eny products.)

\section*{ALPHA MICRO}

Professional Management Servicen
724 Arastradero Rd. 109
Plo Alto, CA 94306
(415) 858-2218

Sierra Computer Co
617 Mark NE
Albuquerque, NM 87123

\section*{APPLE}

IDPC Company
P. O. Bax 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) 451-2905

MicroMotion
12077 Wilshire Blvd. 506
Los Angeles, CA 90025
(213) B21-4340

CROSS COMPLEERS
Nautilus Systema
P.O. Box 1098

Santa Cruz, CA 95061
(408) 475-7461
polyFORTH
FORTH, Inc.
2309 Pacific Coast Hwy.
Hermose Beach, CA 90254
(213) 372-8493

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

M\& B Design
820 Sweetbay Orive
Sunnyvale, CA 94006
Micropolis
Shaw Labs, Ltd.
P. O. Box 3471

Hayward, CA 96540
(415) 276-6050

\section*{North Ster}

The Software Works, Inc.
P. O. Box 4386

Mountain View, CA 94040
(408) 736-4938

PDP-11
Laboratory Software Syateme, Inc.
3634 Mandeville Ceryon Rd.
Loe Angeles, CA 90049
(213) 472-6995

Consumer Computers
8907 LeMese Blvd.
LaMesa, CA 92041
(714) 698-8088

Software Federetion
44 University Dr.
Arlington Heights, IL e000
(312) 259-1355

Technical Producta Co.
P. O. Box 12985

Gainaville, FL 32604
(904) 372-8439

Tom Zimmor
292 Falceto Dr.
Milpites, CA 95035
1802
FSS
P. O. Box 8403

Austin, TX 78712
(512) 477-2207

6000 \& 6009
Kenyon Micronyaterne
1927 Curtis Avenue
Redondo Beach, CA 90278
(213) 376-9941

TRS-80
The Micro Works
P. O. Box 1110

Del Mar, CA 92014
(714) 942-2400

Miller Microcomputer Servicee
61 Lake Shore Rd.
Natick, MA 01760
(617) 653-6136

The Software Farm
P. O. Box 2304

Reston, VA 22090
Sirius Systems
7528 Oek Ridge Hwy.
Knoxville, TN 37921
(615) 693-6583

\section*{6502}

Eric C. Rennke
540 5. Rench View Circle \(\$ 62\) Anaheim Hills, CA 92007

Satum Software, Ltd
P. O. Box 397

New Westminister, BC
VK 4Y7CANADA
00so/Zzo/CPM
Laboratory Miersoyateme
4147 Beethoven St.
Los Angeles, CA 9006 (213) 390-9292

Timin Engineering Co.
9575 Coneese Ave. FE-2
Sen Diego, CA 92121
(714) 455-9008

Appllicution Packegne InnoSys
2150 Shattuck Avenue
Berkeley, CA 94704
(415) 843-8114

Decision Remources Corp 28203 Ridgefem Ct. Rencho Pelo Verde, CA 90274 (213) 377-3533

60001
Emperical Rea. Crp.
P. O. Box 1176

Milton, WA 98354
(206) 631-4855

Firrmware, Boarde and Mechinse
Datricon
7911 NE 33rd Dr.
Portland, OR 97211
(503) 284-8277

Forward Tectnology
2595 Martin Avenue
Senta Clara, CA 95050
(408) 293-8993

Rockwell International
Microelectronics Devices
P.O. Box 3669

Anaheim, CA 92803
(714) 632-2862

Zendex Corp.
6398 Dougherty Rd.
Dublin, CA 94566
Variety of FORTH Product:
Interective Computer Syateme, Inc.
6403 Di Marco Rd.
Tampa, FL 33614
Mountain View Preen
P. O. Box 4656

Mountain View, CA 94040
(415) 961-4103

Supersoft Associates
P.O. Box 1628

Champaign, IL 61820
(217) 359-2112

\section*{Consiltents}

Creative Solutions, Inc.
4801 Randolph Rd.
Rockville, MD 20852
Dave Boulton
S81 Oakridge Dr.
Redwood City, CA 94062
(415) 368-3257

Go FORTH
SO4 Lekemead Way
Redwood City, CA 94062
(415) 366-6124

Inner Access
517K Marine View
Belmont, CA 94002
(415) 591-8295

John S. James
P. O. B0x 348

Berkeley, CA 94701
Laxen \& Harris, inc.
24301 Southland Drive, 303
Hayward, CA 94545
(415) 887-2894

Microsystems, Inc.
2500 E. Foothill Blvd., 102
Pasedena, CA 91107
(213) 577-1471


\section*{FINTH CIITETSOIDIS}

Published by Forth Interest Group
Volume III No. 6
March/April 1982
Publisher
Editor
Roy C. Martens Leo Brodie

Editorial Review Board
Bill Ragadale
Dave Boulton
Kim Harris
Jahn James
Dave Kilbridge
Henry Laxen
George Maverick
Bob Smith
John Bumgarner
Gary Feierbach
Bob Berkey
FORTH DIMENSIONS solicits editorial material, commente and letters. No responsibility is assumed for accuracy of material submitted. MOST MATERIAL PUBLISHED BY THE FORTH INTEREST GROUP IS \(\mathbb{N}\) THE PUBLIC DOMAIN. Information in FORTH DIMENSIONS may be reproduced with credit given to the author and the Forth Interest Group.

Subscription to FORTH DIMENSIONS is free with membership in the Forth Interest Group at \(\$ 15.00\) per year ( \(\$ 27.00\) foreign air). For membership, change of address and/or to abmit material, the address is:

Forth Interest Group
P.O. Box 1105

San Carlos, CA 94070

\section*{HSTORICAL 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, espec ially for observatory automation.

Mr. Moore and several associates formed FORTH, Inc. in 1973 fo 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 Northem Callfomia. Our membership is over 3,500 worldwide. It was formed in 1978 by FORTH programmers to encourage use of the language by the interchange of ideas through seminars and publications.

\section*{PUBLIC NOTICE}

Although the FORTH Interest Group specifies all ite publicatims are non-copyright (public domain), several exceptions exist. As a matter of record, we would like to note that the copyright has been retained on the 6809 Assembly listing by Tatbott Microsysterns and the Alpha-Micro Assembly listing by Robert Berkey. Several conference papers have had copyright reserved. The general statement by FIG camot be taken an absolute, where the author states otherwise.

\section*{FROM THE EDITOR}

Hi! I'm happy to say that starting with this issue, I'll be serving as regular editor of FOR TH Dimensions. I'd like to thank Carl Street, the previous editor, who has been a great help to me during the transition. Carl has made several important contributions to FORTH Dimensions, such as the writer's kit for helping you submit articles. Carl will rejoin FORTH Dimensions as our advertising director beginning later this year.

I'd also like to thank Roy Martens, the publisher, for suggesting that I take the editor's post, and for teaching me some of the facts of life in magazine publication.

I hope to make this magazine as useful as possible to the greatest number of people. Since most of our readers are still learning FORTH at one level or another, I intend to encourage the publication of tutorials (such as Henry Laxen's excellent series which continues with this issue), application stories (sure, FORTH is fun, but let's show the world what we can do with it!), examples of well-written FORTH code (the best way to learn style is by reading elegant examples), and any ideas, discoveries, impressions or feelings you care to express (this is your magazine, after all!).

In short, we'll be concentrating on how to use FORTH in solving problems.

By contrast, system implementation details are more the responsibility of the individual vendors' documentation. In addition, the FORTH community boasts two organizations devoted to improving and extending the language: the Standards Team and the FORTH Modification Laboratory (FORML). Each of these groups convenes annually, and the proceedings of these conventions (available through FIG) are extremely valuable documents for the advanced study of FORTH.

I'm looking to each of you to help make this the kind of magazine you want it to be, by contributing articles, examples, and letters. We don't have a staff of writers, so everything we print comes from you. (If you want to contribute but don't know what or how, drop me a line. I'll send you the information kit that Carl put together, and answer any questions you may have.)

I hope you enjoy FORTH Dimensions. And remember, I hope to hear from all of you.

\section*{Leo Brodie}

\section*{NEW POLICY}

The 79-Standard has been voted on and adopted to serve as a common denominator for transportable FORTH code and for future discussion of FORTH systems. Beginning with the next iasue, FORTH DIMENSIONS will give preference to articles that adopt the 79-Standard

Listings which u8: words that are not 79-Standard are welcome, but if possible explain such words in a brief glossary with a note that they are not 79-Standard. For instance, if your application addresses the name field of a definition (which is illegal in the Standard), you should supply a glossary description of NFA.

If possible, also include the definition of such a word. High level source is preferred, but if necessary, the definition may be written In assembler.

We hope this policy will encourage unification, eliminate ambiguity, and simplify explanations.

\section*{LETTERS}

\section*{FORTH Application Library}

Dear fig,
As distributors in the UK for FORTH : \(า\)., with a rapidly growing customer ase, we are potentially interested in any spplication software that is generally se ful.

Most of our customers are in the prozess control/industrial/scientific sectors wich, by their nature, require fairly spezalized and customized software. Never:heless, we are sure there are many areas \(z^{f}\) commonly useful software and that such software would be useful even if only zs a starting point or guideline, in order to avoid too much reinvention of the wheel:

Such software might be offered as free and unsupported, at media cost, or as a zาargeable product. Whichever way, it reeds to have at least some documenta:.on, (i.e., overview and glossary) but it zoes not have to be a professional packạe.

We have an initial enquiry from a user who needs a 3-term controller program for servo control, and some process mathe--atics for numerical filtering and linear zonversion. As he said to us, "surely someone has done this before and written : up enough to be useful?". So can you elp? If you're offering something free, serhaps we can do a trade for something ou would like.

If people are interested in application exchanging we would be happy to act as a node' for making contacts. And where someone has some software that has a narketable value, we are interested in vobing to create and promote viable zackages. We'll not make any firmer plans =- suggestions until we hear from you!

Nic Vine
Director
COMSOL
Treway House
Hanworth Lane
Chertsey, Surrey KT16 9LA

\section*{Benchmark Battles}

\section*{Dear Fig:}

I believe that the primary considera\(\therefore\) ion of an implementation be fluency of se, and not speed or size except when specific problems arise. But after reading :רe "Product Review" in FORTH DimenEions III/l, page 11 and seeing some benchmarks, I couldn't resist trying the same on my own home-brew implementa:ion: \(4 \mathrm{mHz} \mathrm{Z}-80, \mathrm{~S}-100\) bus (one wait state on all memory ref's). These are the -esults I got, plus another column correctig for my slower clock (but not for the
wait state). I guess I designed for epeed.
Just want to stick up for the ol' Z-80. If other people can brag bout how compact their implementatione are, can't I brag about how fast mine is?
\begin{tabular}{lcr} 
& Timin & Duncen \\
LOOPTEST & 2.3 & 2.9 \\
-TEST & 5.9 & 7.4 \\
*TEST & 44.0 & 54.9 \\
/TEST & 74.3 & 88.6 \\
& & \\
& Bonadio & 4761 \\
LOOPTEST & 1.7 & 1.1 \\
-TEST & 6.8 & 4.5 \\
\#TEST & 17.5 & 11.7 \\
/TEST & 29.4 & 19.6
\end{tabular}

Note
All times in seconds. Each test involves 32767 iterations.

No, I don't use any special hardware. Just the normal Z-80 inatruction set. That mulitply threw me off when I first timed it, but the cycles add up about right. I just can't figure out why everyone else is so slow.

I don't have mass storage. That's why I skipped the last two benchmarks. I store everything in EPROMs. Much faster than those clumsy mechanical devices.

\section*{Allan Bonadio \\ 1521 Acton St. Berkeley, CA 94702}

\section*{Editor's Note:}

Here is the code for the benchmarks published in Volume III, No. 1:
: LOOPTEST
7FFF O DO LOOP ;
: -TEST
7FFF ODO I DUP - DROP LOOP;
: *TEST
7FFF ODO IDUP • OROP LOOP;
: /TEST
7FFF ODO 7FFF 1 / DROP LOOP;

\section*{To "G" or not to \({ }^{\text {" }} \mathrm{G}^{\boldsymbol{m}}\)}

Dear Fig,
I would like to comment on the "Starting FORTH Editor." The "M" commend is bad for reasons of safety and philosophy. It takes a line from the current screen, and puts it "out there" somewhere. If it goes to the wrong place (these things heppen), good luck finding it.

A far better alternative is the inverse command, which I call "G" for "get." G takes the same parameters as \(M\) (block/ line-) and gets a line onto the current screen. I believe that only the acreen
being edited should change. \(M\) violates this rule, \(G\) does not.

One further point: \(G\) inserts the now line at the current line, not under it. This allows you to alter line 0 , which \(M\) cannot.

The next extension is BRING, which gets several lines. It takes (block/line/ count-). I find \(G\) and BRING extremely useful. Comments are solicited.

\section*{Mike Perry}

I sgree: \(G\) is more satisfying from the user's point of view. With M, I find myself checking back and forth between the source and destination blocks repeatedly.

The problem of copying a line onto line zero with " M " reminds me of the sarne problem one has with "U" (also in the "Starting FORTH" editor). I'd like to point out a simple way to "push" a line onto line zero, moving the current line zero and everything else down:

\section*{0 TU This will be the new line zero} 0 TXU

The second phrase swaps lines zero and one.--ed.

\section*{FORTH in its Own Write}

Dear Fig,
The two paragraphs below appeared in en article in BYTE Magazine on pg. 109 of the August 1980 issue. When it first appeared, I agreed with what it was saying but did not feel the need to point it out to others. Now, however, I think that it's time to remind all of us about FORTH and what it isn't. Clearly it isn't any other language.

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

FORTH berefits most from a new, different programming style; techniques blindly carried over from other environments can produce cumbersome results.

It atill eludes me as to why people insist on building things into FOR TH which are "imports" from other languege structures and that in most places do not have any logical place in FORTH. Surely they would not be used by a good FORTH programmer. Take as a simple example spacings. FORTH does not impose indentation or strict spacing requirements as do some other constructs, so why do people insist on indenting? I disagree that this contributes to the readability of the language as FORTH is one of the most terse constructs in existence. One might say that a first attempt to improve the readability of FOR TH should center around removing the cryptological do-dads that are used. For instance, "(a)" should be renamed "FETCH". Likewise, " : " should be renamed "STORE" and "." changed to "PRINT".

Obviously this is absurd and so is the notion of indentation and other pseudo spacing requirements that some say contribute to "good prograrnming style." Good programming style is writing clear, concise, fast code that does simple things and then using that and other code to construct more complex definitions. This is the premise upon which FORTH was based. I have seen readable code that was sloppily written, too big for the job that it attempted to accomplish and in a single word was abominable. However, it "looked neat and clean."

When the FORTH 79 standard was released I applauded. We are all aware of the small ambiguities and possible deficiencies in the standard. However, the standards team must be commended merely because they exist and they at least attempted to create a standard of some kind. Why then don't people write in standard code? It aggravates me to see code in your journal prafixed or post-addended by a phrase similar to "all you need to do to bring this code up to the standard is..." Why not write standard code in the first place?

This letter is purposely provocative and I sincerely hope that you decide to publish it. Through it I hope to force a reevaluation of the way some individuals look at FORTH. Some of us still think that FORTH is elegent because of its simplicity. It is unfortunate that many refuse to see FORTH as the beautiful language that it is, but see it only as another language that they'd like to resemble.

\section*{J.T. Currie, Jr. Virginia Polytechnic lnstitute Blacksburg, VA 24061}

Well-expressed, on both points! Regarding the use of the 79-Standard, see our "New Policy" at the front of this issue.--ed.

\section*{Minneeota Chepter}

Dear fig,
Greetings from the Frozen Wasteland!
This letter is to inform you of the formation of a Minnesota chapter of the FORTH Interest Group. We have had two meetings so far, with attendances of twelve and sixteen respectively. We plan to be meeting once a month. Anyone who is interested should get in contact with us first at the above address.

We hope to start some kind of newsletter in the near future. I've heard that it's possible to get copies of program listings and other handouts which have appeared at Northem California meetings. Could you please let us know how we go about getting copies? I have enclosed a SASE for you to respond.

One of our members is running a Conference Tree (a Flagship for The CommuiTree Group) which we hope to use for interchange of ideas, programs, etc. outside the general meeting, and to complement the newsletter. The phone number for that Tree is (612) 227-0307. The FORTH branch is very sparse right now, however, since we are just getting off the ground.

We are also contacting local computer groups about jointly sponsoring FORTH tutorials for specific machines, and providing a public-domain, tum-key FORTH system that will tum on their machines. We currently have such software for the Apple II, SYM-1, are close on an Osborne1, close on an OSI, and are seeking out a TRS-80 version.

Well, that's our plans for the next few months. We would appreciate your current mailing list of Minnesota residents ( \(55 \times x \times\) and \(56 \times x \times z\) ip codes, I believe).

Hope to hear from you soon:

\author{
Mark Abbott \\ Fred Olson \\ Co-founders of MNfig
}

Happy to hear sbout your new chapter: Your malling list is on its way. And yes, handouts from the Northern California Chapter meetings are available. Here's how to obtain them:

John Cassady of the Northern California chapter has agreed to serve as a clearinghouse. The Secretary of any FIG Chapter can mail, each month, handouts from his own Chapter's meetings to Mr. Cassady. In return, John will send back one set of all handouts he receives each month, including thase from the Northern California meetings. Even if a local Chapter has no handouts, the Secretary must sent at least a postcard to indicate the Chapter's continued interest. The
local Chapter's Secretary will make the necessary copies to distribute to members of that Chapter.

So, let's aee those handouts from all the Chapters! Write to:

\section*{John Cassady}

339 15th Street
Oakland, CA 94612

\section*{Brain-System}

\section*{Dear fig,}

The special FORTH issue of Dr. Dobb's Journal made a deep impression on me and on my son. My son is since 12 years a system programmer and knows more than a dozen computer programming languages. Iam a logician and engineer, code designer and the developer of the only existing proto-model of Interdisciplinary Unified Science and its computercompatible language, the UNICODE.

Thus, I represent a radically different path of scientific development--disregarded by many because it does not promise immediate financial returns.

My approach is centered on a new and far more encompassing system-idea of the temporary name "brain-system" having a physical-hetero-categorical genetically ordered sequence of models of logic. This sequence has a specific case for presentday formal logic and a corresponding simplified variant of the system-idea: this is the system-idea of the digital computer.

UNICODE is the first' specific brainsystem programming language. It is \(s\) content oriented language, it has powerful semantics and register-techniques. It has "words" which are at the same time total programs for the generation of the invars and "content" the term intends to communicate.

I think to study UNICODE will lead to unsuspected breakthrough in the development of programming, especially if thinking has been made elastic and modular by studying FORTH.

1 would like to receive the private addresses of a few creative FORTH fans. In the hope of your early reply, I remain...

Prof. Dipl. Ing. D.L. Szekely P.O. Box 1364

91013 Jerusaiem, Israel
December 1981

Anyone follow that?--ed.

\section*{TECHNOTES}

\section*{ENCLOSE Cerrection} for 6502

\section*{Andy Bigge}

41, Lode Way
Haddenham
Ely, Cambs
CB6 3UL England

On converting my 6502 fig-FORTH (V1.1) to work with 256 byte disc sectors, I discovered (after many system hang-ups) that WFR's 'ENCLOSE' primitive is not guaranteed to work with disc sector sizes greater than or equal to 256 bytes in size.

In his 'ENCLOSE,' Bill uses the 6502 Y register to index through the input text stream, but this register is only 8 bits, \(s 0\) if the text stream contains a block of delimiter characters, e.g., 'space' bigger than 256, it will loop forever, as I found to my cost:

When will this occur? Never from the terminal input buffer, which is only BO characters long.

With a disc sector size of 256 or bigger, if you have an entire sector of spaces in a load screen, then the load will hang up on this chunk of spaces.
or...

If your sector size is bigger than 256, then any chunk of spaces 256 or bigger will hang it.

I encountered this because I decided to emulate John James' method used on the PDP-11 version, where R/W' handles IK every time, so as far as BLOCK, BUFFER, and ENCLOSE are concerned, the disc block is 1024 bytes, and compiling hung up on any text gap bigger than 256 bytes:

Anyway, I ENCLOSE (ha ha) a reviaed version of the ENCLOSE primitive which I am now using, which has full 16 bit indexing. I'm sure some assembly language programmer could produce a neater version, but at least I know that this one works.

Keep up the good work.
By the way, I'm willing to act as a fig software exchange/library in the UK, unless there is someone already doing it?
-ENCLOSE• PRIMITIVE FOR 6SF2 UITM 16-EIT : NDEVINE
THE 'V' REGISTCR SORMS THE LOV INDE: :YTF SIACK LOCATIOL SZ, X FORMS تIF HIGH IN: \(M\) ROTT



\section*{TRANSIENT DEFINTIONS}

Phillip Wasson
Editor's Note: This article appeared in the last issue, but, unfortunately, without the source code. Here is the article as it should have appeared. Our apologies.

These utiliites 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 ares" 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. OISPOSE 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 figFORTH definitions.
```

FIRST 1000 - CDNSTANT TAREA ( Transient area addrese )
VARIABLE TP TAREA TP ! (Tranmient Pointer )
; TRANSIENT ( --- ADDFF)
HERE TP © DP ! ;

- PEKMANENT ( ADDK -.- )
HERE TP ! DP ! ;
: DISFOGE ( --- )
TAREA TP! VOC-LINK
BECIN DUP
EEGIN DUP TAREA US UNTIL DUP ROT I DUP O=
LNNTIL DRDP VOC-LINK
BEGIN DUP 4 -
BEGIN DUP
BEGIN PFA LFA DLP TAREA U\
UNTIL. DUP ROT PFA LFA ! DUP Om
UNTIL DROP DUP O=
UNTIL DROP [COMPILE FORTH DEFINITIONS :
(Example)
TRANSIENT
: CASE ... '
; OF !., '
: ENLCIF ... !
: ENDCASE ... :
PERMANENT
; DEMO1
... CASE
... OF ... ENDOF
... OF ... ENDOF
ENDCASE :
TRANSIENT
- EQUATE (N -..-)
CREATE , IMMEDIATE
DOES) STATE
IF CCOMPILE LITERAL THEN
7 EQUATE SOME-LONC-HORD-NAME
PERMANENT
| DEMO2 ( SOME-LONG-WORD-NAME is comPiled)
GOME-LONG-WORD-NAME . ; ( as a literal )
DIGPOSE ( Removes the words ECUATE, BOME-LONG-WORD-NAME, )
( CASE, DF, ENDOF, and ENDCASE from the )
(dictionary.)
DEMO2 }7\mathrm{ OK (Test DEMO2, it prints e Eeven. )

```

\section*{RENEW TODAY!}

\author{
NOVA bugs \\ John K. Gotwala \\ Computer Technology Department \\ South Campus Courts C \\ Purdue University \\ W. Lafayette, IN 47907
}

I have just finished installing figFORTH on my NOVA 1200, using the listing I received from fig. Instead of running it standalone, as the fig listing does, I run it as a task under RDOS Rev. 5.00.

So far 1 have found four bugs or omissions in the listing. They are as follows:

Page 10 of the listing - EMIT does not increment OUT.
[COMPILE] does not work properly. It can be fixed by removing CFA, from line 07 on page 42 of the listing.

VOCABULARY does not work properly. This can be fixed by adding CFA between AT and COMMA on line 53 of page 44.
(FLUSH) can not be accessed until a missing 〈S1> is inserted after FLUSH on line 13 of page 52 .

After installing fig FORTH, I entered the CYBOS editor from the keyboard and used this editor to boot the fig editor listed in the installation manual. After this experience, I am somewhat pessimistic about FORTH's portability between word and byte addressing machines. I had to make quite a few changes before the fig editor would run. Some examples:

BLANKS expects a word address and ward count.

COUNT expects a word address and returns a byte address.

HOLD and PAD both return word addresses.

If any RDOS NOVA users would like a copy of my "fig-FORTH," they should feel free to contact me.

RENEW NOWI

RENEW TODAY!

\title{
FORTH Standarde Cormer
}

Robert L. Smith

\section*{DO, LOOP, and +LOOP}

There have been some complaints about the way that + LOOP is defined in the FORTH-79 Standard. The first obvious problem is that the Standard does not define the action to be taken when the increment \(n\) is equal to zero. Presumably that was either an oversight, or a typographical error. The most likely correction is to treat the \(n=0\) case the same as \(n>0\), since the arithmetic is defined to be two's complement, and for that arithmetic, the sign of 0 may be considered to be positive. I am aware of other possibilities, but they seem to be fairly difficult to implement or explain.

The second point that is mentioned is that the parameter range seems to have a strange asymmetry. When a positive increment is used, the DO-LOOP index I may not reach the specified limit. However when a negative increment is used, the index I may be equal to the specified increment. Users of fig-FORTH systems have pointed out that the fig +LOOP is symmetric in the sense that for either negative or positive increments the limit value is never reached. One may consider that the Standard version terminates when the boundary between the limit \(n\) and \(n-1\) is crossed, whether the increment is positive or negative.

Finally it has been noted that the Standard LOOP and +LOOP depend on signed arithmetic. Many, but not all, FORTH use a modular or circular arithmetic on DO-LOOPs, allowing the index I to directly address memory. The use of I to address memory in a Standard LOOP may result in a non-transportable program unless a certain amount of care is taken. The Standard version is easier to define than one involving circular arithmetic. Note also that the Standard version allows approximately twice the range of most circular loops (such as in fig-FORTH).

The best suggestions for new looping methods can be found in a paper given by Robert Berkey at the recent FORML Conference. The paper is entitled "A Generalized FOR TH Looping Structure." I recommend that readers interested in the topic get a copy of this paper and implement his suggested words. I wqould like to slightly modify his results for the current discussion. Berkey essertially shows a technique for looping in which the increment for +LOOP may alternate between positive and negative values without necessarily terminating the loop. Modular arithmetic is used so that either signed or unsigned use of the index I may be employed. The increment may be any value. The terminating condition is when
the boundary between \(n\) end \(n-1\) (ectumery \(n+1\) in Berkey's paper) is croseed dynmically. The implementation appears to be even more efficient than that deacribed by Brodie and Sanderson ("Division, Retations, and Loops," Rochester Conference, 1981). The only apparent disadventage of the implementation is that the index is computed by addition or aubtraction. A novel feature of Berkey's implementation is that when the word LEAVE is executed, the loop is terminated at that point (i.e., LEAVE actually leaves). Berkey also sugqests that for normal positive incrementing loops that the index range should include the upper limit, in a manner more consistent with other languages as well as typical use in the fig-FORTH INDEX. Finally, he suggests a construct so that a loop may be skipped entirely if a counting parameter is zero.

The work discussed above is of potential interest to future directions in FORTH. It shows that FORTH is still evolving, even though it cannot effect the current Standard.


\section*{Position Wanted}

I am looking for a software engineering position with another compeny that uses FORTH. I would like to work for a firm using FORTH to develop state-of-the-art systems software; specifically, a FORTHbased development and oeprating system environment to compete head on with UNIX.

Brent Hoffman
13533 37th N.E.
Seattle, WA 98125
(206) 363-0642

\section*{9900 Trace}

Hainz F. Lenk
Loowenste iner Ring 17
6501 Woerrstadt
Gerinany
I have had some trouble getting my 9900 FOR TH running.

To ease the finding of errors I wrote a progran to display all important vectors (IP, W, CODE, R, SP) and the first 7 stack contents. Even the stack's yrowing is visible.

I would like to contribute it to yous, s) you can offer it to all \(99(1)\) usurs with \(h\) 100 M or similar board.

It was a great luck for ine that 1 dind not need the addresses \(>37 \mathrm{C}\). and \(>571\)-, anul could use it for a branch to the STA IIS; program. This program is switethed off hy the code HEX 455384 ! and switched on by HEX 457384 :.

The prograin list contains the mutines for terminal input and output, ton.

I hope I can help some panple with my prograrn.




 SEVEN SIIALIK LUNII:NIS
- HEIN/ IENK, LII W.NTIIEINIK KING I/, LMII WEOHHSTDT

GO
FFOO
EE
FF20
CO
FF2
D DAIA fifix, xth 1 V:cithe fl: batn tion
 UAIA FMII!


* Fead data 10 sitali foinitd fir kfi
- LALE WIIH xift EKG. 1

it 1 timathon:d
G1:
\(1+15\)
\(16 F C\)
3506
\(1 E 12\)
360
In \(\because 1\)
JN IN inllir

1111:1 n!it:1
hatilivilk INIK Dent

- CALL WiHI x(H, Hill.
\begin{tabular}{|c|c|}
\hline \multirow[t]{2}{*}{} & \\
\hline & \\
\hline & \\
\hline \multirow[b]{4}{*}{} & \\
\hline \[
\begin{aligned}
& 34 A \\
& .34: \\
& 34 E \\
& \$ 5 \times 1
\end{aligned}
\] & \[
\begin{aligned}
& 406 \\
& \text { CSK5 } \\
& 180 \\
& 10 \mathrm{~m}
\end{aligned}
\] \\
\hline & \\
\hline & \\
\hline & \\
\hline 37C & 247 \\
\hline 372 & Fe \\
\hline 3FM & C2F9 \\
\hline 342 & C178 \\
\hline 384 & 457 \\
\hline
\end{tabular}
-

- SUBPRUGRAM TO OUTFUT A STRING TERMINATED EY >00 - CALL WITH XDF CADFEEB, 3
\(+\)
\(\begin{array}{ll}\text { CE } & 000 \mathrm{~B} \\ \mathrm{CA} & 1302 \\ \text { CC } & \mathrm{CzO2} \\ \text { CE } & 10 F \mathrm{C} \\ \mathrm{jO} & 300\end{array}\)
mixIT

**
- EUEPROGFAM TO OUTFUT A HEX winत
- call xuf sourree, 4

WEMTRY LI K1.4 CDUNT
D2 2014
4
6 CTEB MON \#R11,H11 MOU RII, Fi. ALIGNHENT ANDI R2 HRFIMO MOFY EI 2, QiNi MMEEF? HLE NMM RI, TIMO RDUGI IETTER Am AI \(\mathrm{Hz}, 30 \mathrm{OH}\)

ADsusit ASCII
зім
2CE?
48 MWEXI
\(\square\)
\begin{tabular}{|c|c|c|}
\hline xos & M, 2 & O17 PIT \\
\hline Sike. & F11, 12' & Shlt \\
\hline DEC & H1 & I:MANT-1 \\
\hline JME: & WMEXT & 2EFOU \\
\hline R1w & & ExIt \\
\hline
\end{tabular}
 - PGINT GIAIIRI FHENENM

\begin{tabular}{|c|c|c|c|c|c|c|c|}
\hline F8 & 1904) & SIAlls & NKF. & & & SP/Nr & \\
\hline - \({ }^{\text {A }}\) & ELEO & & Xix & \(\downarrow\) & AH, ? & HSG & 1 It \\
\hline +1: & A \({ }^{\text {a }}\) & & & & & & \\
\hline
\end{tabular}

FF. Cimis movery.NO CISY



109
10 C
10N:
\(110 E\)
116
110
112
112
112
114
116
118
\(11 A\)
\(11 A\)
\(116:\)
115
\(1: 4\)
\(1 \because 2\)
\(1: 4\)

\section*{126 \\ 126
129}

120
\(12 n\)
124
\(1: 4\)
\(1: 25\)
151
139
125
130
132
132
136
130
13
130
139
\(14:\)

mN KII, KI
th.C. Hei
xit kis. 4
xisp Ha. 3 MiU talise
xim 65.4

xum Hill,
xull 4 C:
xim. KE. 1
MW.
(MAPIH 1 SIAM:


STOMA XISP RI, : NLY EI.AN
\(x+1\)
\(x\)
(1) 4 K3. 4

MUS FI.ANH


Decil ms ney sim:
LEC: KI COANII
LEC: KI
HEHEA INIII ASHO

gitxit

- MOI UKICIINAI IOYNANIIC KAM MIGCAIIINN FK
- MOI WiAk WIIII AN UNIEKMINAIED D.


- THF INELK IMIEFREYEK 15 CHANGED TG PFINT ALL - PUINTERS (IF, W, CODEFODY, R, SF') NND STACK.
- THE STATUS IS SWITCIMED ON EY HEX 4.57 ?日, !
- THE SIAIIS is SNIICHED OFF BY HEX 45S 384.

AH*B 37C
\(11 \cdot R 7\), PFB PC OF STATUS
HON LIP+, 2W
MOV EZW+, ZTEFP fRIMT TO. BODY BRANCI IO STATUS

\title{
A TECHNIQUES TUTORIAL: EXECUTION VECTORS
}

\author{
Henry Lexen \\ Laxen \& Harris inc. 24301 Southlend Drive Hayward, CA 94545
}

This month, we continue our exploration of FORTH programming techniques by taking a look at a concept known as Execution Vectors. This is really a fancy name for very simple concept, namely using a variable to hold a pointer to a routine that is to be executed later.

It is only fair to warn you that the dialect of FORTH that I am using is the one discussed in Starting FORTH by Leo Brodie. It has several differences from figFORTH, not the least of which is the fact that in figFOR TH EXECUTE operates on code field addresses (efa's), while in Starting FORTH EXECUTE operates on parameter field addresses (pfa's). This may not seem like a big deal, but if you have ever fed EXECUTE a pfa when it was expecting a cfa, you have undoubtedly remembered the result. Anyway, my EXECUTE uses pfa's. Its function is to ferform or EXECUTE the word that this pfa points to. An example will clear this up. Suppose we have the following:
```

: GREET ." HELLO, HOW ARE YOU";
- GREET (LEAVE THE PFA OF
GREET ON THE STACK)

```
    EXECUTE (AND NOW PERFORM IT)
    the result is:
    hello, how are you
    which is the same result as just typing
GREET.

The above may not seem too significant, but the implications are tremendous. Consider the following examples:

\section*{VARIABLE 'EMIT}
: EMIT (CHAR --)
'EMIT @ EXECUTE;
- (EMIT) 'EMIT :

I assume that (EMIT) is a routine which takes a character from the stack and sends it to the terminal. By defining EMIT to use 'EMIT as an execution vector, we now have the ability to redirect the output of FORTH in any manner we choose. For example, suppose we want all control characters that are sent to the screen to be prefixed with a caret. We could do the following:
```

CONTROL-EMIT ( CHAR --- )
Dup 32 ( BLamk ) < IF
32 "a (EmiT) if (Control Cher?'
64 (ABCIIA-1), ( End convert $t \hat{t}$ )
TIEN
(ENIT):
COMTROL-EMIT •EMIT :

```

Now all regular characters will fail the test, since they will be larger than blanks;
however, control characters will aucceed and will be incremented by 64, making them displayable.

There are several other FORTH words that have proven useful to vector. Some of these include:

KEY input from keyboard primitive
CREATE change header structures
LOAD useful for many utilitios
R/W disk i/o primitive
For example, if LOAD were vectored, then by redefining it to print a screen instead of loading it, you could write a print utility which prints screens in load order by LOADing a load screen and redefining LOAD to print. CREATE could be changed to add the screen number of each tefinition to the dictionary header so that it could later be retrieved with VIEW or the equivalent. KEY may be changed to get its characters from a file somewhere instead of the keyboard. In short, there are a thousand and one uses for Execution Vectors.

But be careful, I may have opened Pandora's box with the above selling job. There is a price to be paid for execution vectors, and that is complexity, the archenemy of reliability. Every word that you decide to vector at least doubles the complexity of the FORTH system you are rurning, since it introduces at least two or more states that the system can be in. You must now also know what the version is of each execution vector you are using. If you have 3 different EMITs and 2 different KEYs and 3 different LOADs, you have a total of 18 different states that the system can be in just on these vectore alone. So use vectors sparingly, otherwise you will lose control of the complexity very very quickly.

Heving decided to use execution vectors, we're now faced with different approaches towards implementing them. The one described above works, and is used by many people, but it has one unfortunate property, namely the need to name a variable which is basically overhesd. Here is another way to accomplish the same thing without having to define a variable. Consider the following:

```

: ExECUTEI
CREAFE (--)
I'I DIN:
a EXgCUTE :
1! 1P, (PFA ---

```
: ExECUTE:
 a EXECUTE

DIE is used to send an orror meange to the terminal and reset the FORTH eriter into a clean state. EXECUTE: is o cofining word which initializes itsolf to DRE, but hopefully will be chenged latar by the user. Words defined with EXECUTE: cm be changed with IS as follows:

\section*{EXECUTE: EMIT}

\section*{- (EMIT) IS EMIT (or perhaps) \\ - CONTROL-EMIT IS EMIT}

What EXECUTE: has done is combined the variable name with the Execution Vector name into one name. IS is used as a convenience, so that the user cen forget the internal structure of words defined by EXECUTE:. Also it provides an extremely readable way of redefining Execution Vectors. Notice that as defined, IS may only be used during interpretation. I leave it as en exarcise for the reader to define an IS that may be compiled within : definitions.

Another approseh to redefining execution vectors is via the word ASSIGN. It could be defined as follows:
```

- (MBEIEN) (CFA --- )
R> 2+ 8ump !
agsien (--- ,
CONPILE (NASIEN)
[. CFA J LITEAAL. I IMHEDIATE
It would be used as follows:

```
```

UPPER-CNMY ( --- )

```
UPPER-CNMY ( --- )
    C] EMIT AESIGN
    C] EMIT AESIGN
        Dup %% ( AECII a-1), IF
        Dup %% ( AECII a-1), IF
            DUP 123 (ABCII }x+1\mathrm{ ), \F
            DUP 123 (ABCII }x+1\mathrm{ ), \F
            THEN
            THEN
        HEN
        HEN
        (EMIT) ( as ALMAYS )
```

        (EMIT) ( as ALMAYS )
    ```

When UPPER-ONLY is executed, EMIT is redefined to execute the code following the ASSIGN, which will convert all lower case characters to upper case, and send them to the terminal. Note that unlike IS, ASSIGN may only be used within : definitions.

That's all for now, good luck, and may the FORTH be with you.

\title{
CHARLES MOORE'S BASIC COMPILER REVISITED
}

\author{
Micheel Perry
}

In this paper I will discuss several interesting features of the "BASIC Compiler in FORTH" by Charles Moore (198i FORML Proceedings).

Why is a BASIC compiler interesting? There are a number of reasons. Foremost of them is that BASIC is in many ways typical of a variety of popular languages, particularly FORTRAN, PASCAL, and ADA. Conspicuous features of these languages are algebraic notation, lack of access to the underlying hardware, poor input and output facilities, and non-extensibility. FORTRAN and BASIC also suffer from poor structuring due to the extensive use of GOTO. These languages all tend to be best at solving equations. Other prominent features of BASIC are it s use of statement numbers as labels, low speed, and its use of a few complicated functions (e.g., PRINT) rather than many simple ones.

Why is it slow? BASIC interpreters usually convert source code statements to an intermediate form, where keywords become tokens. The token interpreter is slow because tokens must be deciphered (translated into actions) at run time. This BASIC to FORTH compiler produces code which runs unusually fast. This is because it produces FORTH object code, i.e., sequences of addresses of code routines.

You should look at the example programs (blocks 80-82) before reading the text. You will notice that each BASIC program becomes a FORTH word named RUN. It is executed by typing its name, i.e., RUN. This is how BASIC usually works; you type RUN to execute the program. It serves to demonstrate that from FORTH's point of view, BASIC only knows one "word," RUN. Is it not more use ful and flexible to let routines have any name, and to be able to execute any of them by typing its name? Yes, and that is a key feature of FORTH.

\section*{How it Works}

I will refrain from commenting on the intrinsic value of a BASIC compiler; that has already been covered well in Moore's paper. The principal features I will diecuss are the handling of operator precedence, variables in algebraic equations, and the use of the FORTH compiler. The most important part of this BASIC compiler is its ability to convert algebraic (infix) source code to reverse polish (postfix) object code.

A BASIC program is compiled intide the colon definition of a word named RUN. This means that the FORTH system is in its compile state, and any words to be
executed during compliation must be immediate. This use of the FORTH compiler was perhaps my greatest lesson from studying this BASIC compler. The ordinary FORTH compiler is far more vereatile than I had realized. If I had written this compiler, it would doubtless have run in the execution state and would have been far more complicated as a reoult.

Let's look at en example. The BASIC statement
10 LET \(X=A+B\)
will be compiled into object code equivalent to the FORTH expresaion \(X A\) @ \(\mathrm{O}_{\mathrm{O}}+\mathrm{SWAP}\) !
where \(X, A\), and \(B\) are variables. Ono of the variables ( \(X\) ) returne on address, the rest retum values (with a fetch). The add is compiled after the fetches of the values to be added. The equals becames the "SWAP!" at the end. Becaute the source code (in BASIC) is in algebraic notation, and the (FORTH) object code is in reverse polish order, some way is needed to change the order of operations when compiling the BASIC program. The mechenism which controls the compilation order is based on the idea of operstor precedence, which means that some operators are asaigned higher priority then others.

\section*{PRECEDENCE}

The idea of operator precedence is a prominent feature of most computer langueges (FORTH is a noteble exception). Operations are not necesearily performed in the order you epecity. An example will help. The equation \(X=5+7\) - 2 could mean either \(X=(5+7) * 2\) or \(X=5+\) (7*2), usually the latter. In FORTH this would be \(72 * 5+X\) : where the order is explicit. In algebraic lenguages some method is needed to clarify the order of evaluation of operators in exprestions. That is what precedence does. Each operation is assigned a precedence level. Operations with higher precedence are performed earlier.

During compilation of the BASIC program (the FORTH word named RUND the compliation of many words is deferred. This allows the order of words to differ between the source code and the object code. Take ' + ' as en example. To defor compllation of ' + ' new word is created which is immediate (and 80 executes at compile time). When thin new word is executed, it leaves the addrees of ' + ' on the atack, and on top it leaves the pracedance value of ' + '. The defining word PRECEDENCE creates the new word as follows: "2 PRECEDENCE +" . This creates a new, immediate word named ' + ', which will leave the sddrese of the old word ' + ' under the value 2 .

The word which decides how long to defer compilation is DEFER. DEFER looks at two pairs of numbers on the stack. Each pair consiats of an address and a precedence value. If the precedence of the top pair is larger then that of the lower, DEFER does nothing. If the top precedence is lese than or equal to the one below, the addrese part of the lower pair is compiled, and its precedence is discarded. DEFER will continue to compile until the upper precedence is larger than the lower.

So how do you get sterted? Essentially, most BASIC keywords (such as LET) execute START wahich leaves NOTHING 0 on the stack, where NOTHING is the address of a do nothing routine and 0 is its precedence. This pair will remain on the stack during the compllation of that statement, because everything has higher than zero precedence.

At the end of each line, RPN is execut ed. It performs a 01 DEFER, which forces the compilation of any deferred words, because every operator has a precedence of at least 1. RPN then consumes the 0 and executes NOTHING. Actually, each statement is ended by the start of the next. BASIC keywords such as LET execute STATEMENT, which contains RPN (to finith the provious statement) and START (to begin the next).

\section*{BRANCHING}

Three new branching primitives are used. They are compiled by various higher level words. JUMP is used by GOTO. SKIP and JUMP are used by IF-THEN. JUMP is compiled followed by an absolute address. When executed it simply loads that address into the IP (virtual machine instruction pointer). When SKIP executes, it takes a boolean off the stack. If true it adds 4 to the IP, kipping (usually) the following JUMP.
(NEXT) is used for FOR-NEXT loops. It complied followed by en mookute addres. When executed it takes three parameters from the atack: final value of the loop index, step size, and the address of the variable containing the current value of the loop index. It adds the step (plus or minus) to the variable, and loops until the index passes the limit.

Adding COSUB would require another brenching primitive, CALL.

\section*{STATEMENT NUMBERS}

Each BASIC statement must be preceded by a number. This number acts as a label, allowing branches between lines. In this compiler, the numerical value of the labels does not affect execution order. When a statement number is encountered, it is compiled in line as a literal. The address of LIT is compiled followed by the literal value 10 . For example, when the statment " 10 REM" is encountered, 10 is compiled as a literal. The keyword REM is immediate, and so is executed. It begins by executing STATEMENT, which, amongst other things, fetches the value of the line number just compiled (10), and enters it into the statement number table (ilS) along with the address (HERE) of the start of that statement. STATEMENT then de-allocates the space used by the literal :0 (with a -4 ALLOT). It scans the table and resolves any forward references to the new statement. When a forward reference occurs, as in "GOTO 50" before statement 50 is compiled, GOTO compiles 'JUMP 0'. The zero will later be replaced by the address of line 50. The reference is entered into the table with the address to he patched instead of the actual address of statement 50. Additional forward references to the same point will be chained to each other. To indicate that this is a forward reference, the address in the table is negated. This means that 9ASIC programs must be compiled below 9 COCH , so that all addresses appear to be dositive. Here simplicity was chosen over zenerality.

\section*{VARIABLES}

There are two particularly interesting things to notice about variables. They are immediate, and they know which side of an equation they are on. Three types of variables are supported: integers, arrays, and two dimensional arrays. Variables must be declared (defined) before use. The BASIC expressions: LET \(X=A+B\) (where \(X\), \(A\), and \(B\) are variables) compiles into the following FORTH equivalent: xA@5@+SWAP:
Notice that when an integer appears on the left of an equals sign, it must compile its address, and when on the right side, its value (address, fetch). Also note that only one can appear on the left, while many can be on the right.

The way this is implemented is surprisingly simple. The variable ADDRESS contains a flag which indicates which side of the equals sign a variable is on. The word LET sets ADDRESS to l. "INTEGER \(X^{\prime \prime}\) creates a variable named \(X\), which is immediate. When \(X\) is executed it compiles its address. \(X\) then examines ADDRESS. If it is true (non-zero), \(x\) simply makes it zero. If ADDRESS is false, \(X\) compiles a a after the address, thereby rturning the value when the BASIC program is run.

Notles that the equal ith in mos role in this procest, overything in key words (e.9., LET) and veri.gime.

\section*{Future Directions}

Many more features can easily be added to this BASIC compiler. But why bother? A much more fruitful line of endeavor would be to make use of the lessons learned in this compler to write compilers for other, more useful, languages such as C. A C complier which is easy to modify and extend, and just as portable as FORTH is, could actually be


\section*{Tramportable Control Structures With Compiler Security}

Marc Perkel
Perkel Software Systems
1636 N. Sherman
Springfield, MO 65803

This article is en enhancement of the idea presented by Kim Haris at the Rochester FORTH Conference (from the Conference Proceedings, page 97). Basically, the article proposes a wordset of primitives for defining control words such as IF , ELSE , THEN, DO , LOOP , BEGIN , WHILE , REPEAT, UNTIL , AGAIN, CASE, etc. Kim points out that these strucures are either compiling a branch to a location not yet defined (such as IF --> THEN ) or back to a location previously defined ( BEGIN <-- UNTIL ). There are two steps in compiling either kind of branch: marking the first place compiled and then later resolving the branch. This observation leads to four of Kim's words:
>MARK Marks the source of forward branch and leaves a gap.
>RESOLVE Resolves forward branch and leaves a gap.
<MARK Marks destination of backward branch.

\section*{<RESOLVE Resolves backward branch.}

I complement Kim at this point for his excellent choice of names. Here's where
compiler security comes in.
The word >RESOLVE is filling a gap left by >MARK. If >RESOLVE were to first check to make sure a gap was there ( DUP @ 0 ?PAIRS ) it would help ensure that the value on the stack was indeed left by >MARK. Likewise, if <RESOLVE made sure that the point where it branches back to does not have a gap (DUP @ NOT 0 ?PAIRS ) it would guarantee that it was not answering a >MARK. This method allows some compiler security where it is important not to carry pairs on the stack.

\section*{Example:}
: 3 MARK HEREO,;
: \(\operatorname{PRESOLVE}\) DUP 0 ?PAIRS HERE SWAP:;
SMARK HERE;
<RESOLVE DUP @ NOT O ?PAIRS, ;
IF C, MMARK;
ENDIF >RESOLVE;
ELSE C3 IF SWAP ENDIF ;
BEGIN <MARK;
UNTIL C, <RESOLVE;
AGAIN C3 UNTIL:
WHILE IF ;
REPEAT SWAP AGAIN ENDIF ;

72
O (Charles Mcore's BASIC sagoiler, coditied for fig-finth )
 2 VOCABULARY LJgIS vOCABULARY IMPUT FDRTH DEFIMIHIOWS

4: hoad RLK \(3+\) LDAD ;
: (5ETi) BL MRRD HERE MLABER DROP :
: (.) 5-:5 SMAP OVER DAB5 : \#5 SISN :);
0 variaple is 128 alloi
: SCR A IS 2+ © \(2!\);

10: ! 93 MCRD ; IMmDiate
II ARITHRTIC DEFIMITIOTS


:1: ; [n] is 1 PRECLDEMCE;
:S forth defintions

\section*{74}
( Branchang - Magh level)
1
? : Jump R) J TR;

: (We:I) ! to : step I variable address -- )
200p + ! : atd step to var !

OC If suap THEN -

(MEXT) CMPILE (ME:T: ;
10
12

15

\section*{76}
( Statement mumers)





HERE MEGATE SMAP ;





73
(Precedence)
0 variane abdeess o variame il

: Mefer (anan-an) Mo +
HEGIK 2OVER SMAP DROP OVER < MOT
mille 2Smap niop CFA, Repeat;

(RULLDS , R), IMREDIATE DOES 23 DEFER ;
: apm (a) O: befer 20RGP il 3 of acort Syntax";
; mothing ;
 aRIt!metic: Immediate


75
(Variables)
: Integer (builds 0, imadiate docs) tcompllej literak adoress a If O address! else compile o then ;

SMAP )R 7 DEFER R) [COUPILE] LITER*
ADORESS a IF O ADDRESS:
ELSE - 7 Ha + 2SMAP TMEN:
: ( 1 ] (1:3-a) Smap 1-2t + ;
 DOES) : [ + ] (ARRAY: ;

 DOES) ' [ \(\mathrm{t}+\mathrm{J}\) (ARAYY):

77
( BasIC)
: LET STAYEment 1 abdatss : : imitidiate
: FDP LCOMPLLEI LET ; IWEDIATE
: :1] COMPILE 1 HERE ;
: TO RPW DRGP , L:I 0 : IMAEDIATE
: STEP RPW DROP : MERE O ; IMEDIATE
: meit statenent 2padp, ciextio 1 abdpess ! : imediaye


: stap statement conplle ;s ; imegiate
: ENE STATEMENT 2DRCP [COMPLLE] ; [CORPILE] FGRTK ; !medIATE
: (GOTO: (EETE) canpile junp cmam, ;
: gote statement (erioi ; Imediate
: IF Statenent lo6ic ; inmediate
: TMEK RPM O COMPILE SKIP (GOTO: ; IWHEDIATE

Michael Perry 190:


\title{
A ROUNDTABLE ON RECURSION
}

Recursion, as it applies to FORTH, is the technique of defining a word in such a way that it calls itself. One of the nicest examples I've seen of a good use for recursion can be found in Douglas \(R\). Hofstadter's book Godel, Escher, Bach. He describes a system which can produce gramatically correct phrases out of parts of speech.

I'll use FORTH to describe his example:
```

F FANCY-NOUN
4 CHOOSE
(select random number 0-3)
CASE
O OF NOUN ENDOF
1 OF
NOUN PRONOUN
VERB FANCY-NOUN ENDOF
2 OF
NOUN PRONOUN
FANCY-NOUN VERB ENDOF
3 OF
NOUN PREPOSITION
FANCY-NOUN ENDOF
ENDCASE;

```

Three of the four possible variations on FANCY-NOUN include a call on FANCYNOUN itself. Case 0 might produce "books." Case 1 might produce "man who reads books." But Case 1 might also oroduce something more complicated, like man who reads books that explain algebra," if the iner call to FANCY-NOUN decides to get fancy.

Normally FOR TH deliberately prevents recursion so that you can call an existing word inside the definition of a new definition of the same name. For example:
: + SHOW-STACK + SHOW-STACK ;
This example might be a redefinition of plus to teach beginners what the stack looks like before and after addition. The olus that is called in the middle of the definition is the original + , not the one yeing defined.

FOR TH prevents recursion with a word called SMUDGE. This word usually toggles a bit in the name field of the word most recently defined. With this bit toggled, the name is "smudged"; that is, unrecognizable. In the definition of + above, the colon lays down a head in the dictionary, and then executes SMUDGE before compling the rest of the definition.

When the second + is encountered, the compiler searches the dictionary for a word of that name. The new head with the same name is bypassed only because it has been smudged.

At the end of the definition, semicolon again executes SMUDGE. This toggles the bit back to its original state, so that the name is again findable.

There are various means of circumventing FOR TH's protection against recursion. Here are two recent contributions from our readers:

\author{
A Recuraion Technique \\ Christoph P. Kukulies Aachen, West Germany
}

Here is my solution to the problem of recursion in FORTH shown in a possible way to implement the ACKERMANN's function (see FORTH DIMENSIONS, Vol. [II, No. 3, p. 89).

First test if your FORTH-system is "crash-proof" with the following sequence:
: CRASH [ SMUDGE] CRASH; SMUDGE CRASH

After having recovered from CRASH you should try this:
( \(m n \rightarrow\) ACKERMANN ( \(m, n\) )
:ACKERMANN ( \(m n \rightarrow\) ACK)
[ SMUDGE] SWAP DUP 0 = IF DROP 1+
ELSE SWAP DUP
\(0=\) IF OROP \(1-1\) ACKERMANN
ELSE OVER SWAP
1-ACKERMANN SWAP
1-SWAP ACKERMANN THEN
THEN ; SMUDGE
Be aware of typing
34 ACKERMANN.

\section*{Another Recursion}

Arthur J. Smith
Osahawa Canada LIG 6P7

Regarding the recursion problem, I think that I have found a more elegant solution. The solution involves an SMUDGE the word being defined.

I define a word RECURS as follows:
: RECURS SMUDGE; IMMEDIATE
then use the word to bracket the recursive self definition as in the example:
```

: SUM
DUP 1- DUP IF RECURS SUM RECURS ENDIF
;

```

I use the RECURS word in tree searches.

\section*{Editor's note:}

The technique that is generally preferred was deacribed by Joel Petersen in the original article. It defines MYSELF as
: MYSELF
LATEST PFA CFA, ; IMMEDIATE
or, for some other versions such as polyFORTH:
: MYSELF
LAST @ @ \(2+\); ; IMMEDIATE
MYSELF simply compiles the code field of the latest header in the dictionary (the word being defined) into the definition.

The problem with using the word SMMDGE inside a definition is 1) it's not readable, since smudging has nothing to do with what the definition is about, and 2) its behavior is different on different systems.

Similarly, having to say RECURS ACKERMANN RECURS is not quite as readable as simply MYSELF.

An even more readable solution is this:
: : R
[COMPILE] : SMUDGE; IMMEDIATE : R;

SMUDGE [COMPILE]; ; IMMEDIATE
Here a special version of colon and of semi-colon named \(: R\) and \(R\); are defined to allow recursion without any other hoopla.

RENEW

RENEW TODAYI

\title{
8080 ASSEMBLER
}

\author{
John J. Caseady \\ 339 15th Street \\ Oeklend, CA 94612
}

This 8080 assembler has been available in a slightly different form for approximately one and one-half years. It appears to be bug-free.

ENDIF 's have been replaced by THEN, and AGAIN has been removed in conformance with FORTH-79. I have never had occasion to use AGAIN ; 1 doubt if I'll miss it.

I have removed the compiler security. We frequently want non-structured control mechanisms at the code level. The ?PAIRS really gets in the way.

I have introduced three macros: NEXT PSH1 and PSH2. They emplace, respectively, a jump to NEXT, a jump to the by te before NEXT and a jump to two bytes before NEXT. Literally, PSH1 means push one leve! (HL) and fall into NEXT. 1 believe this is a more traditional approach and the source code has a cleaner appearance.

The actual address of NEXT is stored in (NEXT) . Its value is plucked from ; S . This technique was suggested by Patrick Swayne of the Heath User's Group. I say "suggested" because Swayne's method is a bit different.

1 have left out the conditional CALLs. I never used them and they can always be " C , " 'd in. The conditional jumps are, of course, handled automatically by the conditionals: IF WHILE and UNTIL, in conjunction with the flag testers: \(0=\) CSPE \(0<\) and NOT .

1 have opted to retain the immediate instructions MVI and LVI as opposed to an immediate flag \#.

The 1 MI 2 MI etc stands for "number one machine instruction" etc. The first cut of this assembler was written when three letter names were the craze.

I have a selfish motive in publishing this assembler. I hope that this will flush out assemblers for other processors and that there will be a "rush to publish." There is a good reason to do this besides vanity. If someone else publishes the assembler for the "xyz" chip that you use, and it becomes established, it means that you will have to change your code to conform with the quirks of the "established" version. It pays to get there first.

```

Screen 51 33H
( EXAMPLES USING FORTH }8080\mathrm{ ASSEMBLER 1 81AUG17 JJC 80MAR12 )
FORTH DEFINITIONS HEX
CODE CSWAP ( WORD-1--- SWAPS HI AND LON BYTE OF WORD ON STACK )
H POP L A MOV H L MOV A H MOV PSH1 C;
CODE LCFOLD (FROM-2 QTY-1--- CONVERTS LOWER CASE TO UPPER )
D POP H POP
BEGIN D A MOV E ORA O= NOT
WHILE M A MOV 60 CPI CS NOT
IF 20 SUI A M MOV
THEN D DCX H INX
REPEAT NEXT C;
;S
een 52 34H
( EXAMPLES USING FORTH 8080 ASSEMBLER 2 81AUG17 JJC 80MAR12 )
CODE CMOVE (FROM-3 TO-2 QTY-I--- SAME AS IN NUCLEUS )
C L MOV B H MOV B POP D POP XTHL
BEGIN B A MOV C ORA O= NOT
WHILE M A MOV H INX D STAX D INX B DCX
REPEAT B POP NEXT C;
CODE -CMOVE (FROM-3 TO-2 QTY-1--- SAME BUT OPP DIRECTION )
C L MOV B H MOV B POP XCHG
if ! \& 3 DAD XCHG XTHL B DAD
BHGIN B A MOV C ORA O= NOT
WHILE H DCX M A MOV D DCX D STAX B DCX
REPEAT B POP NEXT C;
: MOVE (FROM-3 T:)-? NEY-1-_- SMART MOVE, DOES NOT OVERLAY)
\R 2DUP RD ROT ROT - THEN
;S
een 53 35H
(EXAMPLES USING FORTH 8080 ASSEMBLER 3 81AUG17 JJC 80MAR12)
80 CONSTANT CMMD ( COMMAND BYTE)
FO CONSTANT CMMDPORT ( COMMAND PORT )
F1 CONSTANT STATUSPORT (STATUS PORT)
LABEL DELAY ( --- DELAY CONSTANT IN DE, DON'T USE THE STACK )
BEGIN D DCX D A MOV E ORA 0= UNTIL RET C;'MASK-1---)
CODE STATUS
i FOF CMMD A MVI CMMDPORT OUT
1324 ( LX:C JELAY CALL
BEGIN
STATUSPORT IN L ANA 0= NOT
UNTIL NEXT C;
;S

```

\section*{Siove of Eratortences} in FORTH

Mitchell E. Timin Timin Engineering Co.

The enclosed version of Eratosthenes \(S\) sve was written for an implementation of Timin FORTH release 3. I was pleased that it executed in 75.9 seconds, as compared to the 85 seconds of figFORTH. Mine was run on a \(4 \mathrm{MHZ} \mathrm{Z}-80\) machine, as were the others in the BYTE magazine article.

The speed improvement is primarily de to the array handling capability of Timin FORTH release 3. FLAGS is created with the defining word STRING; \(n\) LAGS leaves the address of the nth element of FLAGS. This calculation occurs in machine code.
```

SCR 5S
0\& The Sieve of Eratenthenem, After J. Gilbreath, EYTE P/8, )
O EIG0 CONSTANT SIZE SIZE STRING FLAGS ( make arrar of flag=)

```

```

            0 O FLNOS SILE i FiLu which rembims on too of gtack)
            SITE 0 DO (repeat following loom eiq0 times)
        I.FLAGS Ce (reten mext flam to top of stack,
    ```

```

            I DUP - 3, (calculate the meime number,',
    ```

```

            BEGIN DLP SIIE < MHILECK, repent for K< BI90;
                O ovis FLACS c! {ropeat for Kiag;
                    OVER + FLMGs C! (add mrime to K)
            REPEAT
            DFOP DROP 1* ( droek t.mpime. incroment counter)
        ENOIF
    ```

```

- 36
testing the sieve aloorithm, O UNRIAELE KOUNT
OELL }7\mathrm{ ENIT
NEW-LINE CR O OUT | \&
NEW-LINE? OUT E 70 > IF NEN-LINE ENOIF ;
PRIME-TEST BEL (firint sound the bell)
100 D0 PRIFE LOEP BELL (run thempimetinder 10 x I
above if for timime test. belom is for valitation)
o KOUNT | NEN-LINE bolowitefor vailication ciear counter, start new line)
SIZE 0 00 MEN-LNNE (chear counter, stach lias)
I FLACS C (ser if 1t's set)
IF I DOP C 3. (calculate the mrime mumer)
7.R NIN-LINET (dismler it)
1 KOUNT +1)(count it)
LOOP CR KOMNT ? "PRIPIES . ( digmlay the count)

```

\section*{SKEWED SECTORS FOR CP/M}

In regard to Micheel Burton's article in FORTH DIMENSIONS, III/2, pege 53, "Increasing fig-FOR TH Disk Access Speed," I enclose a simple mod to the 8080 or \(\mathbf{Z 8 0}\) assembly list to effect the CP/M kewed sector diak VO. The FORTH routines I used to test the secheme are included. The first cluster or screen is offeet by 52 sectors so that the operating system is transparent and acreens 0 and 1 hold the directory. I move the message screens to SCR\# 24 and 25 leaving 2-20 for the FORTH binary program run by CP/M or CDOS.

In order to check eny increase in diak access speed I timed the following operation with a 10 screen buffer:

\section*{2027010 MCOPY 2027010 MCOPY 2027010 MCOPY}

Elepsed times were 204 and 138 seconds for straight and skewed sectors reapectively. Note that this reflects disk access speed for read/write of several sequential sectors and in no way compensates for inadequate plaming or poor programming in other disk I/O epplications.

If this seems trivial, then you have no need for CP/M file compatible VO. My motive for these changes is the desire to write the assembler program for figFORTH via modem (easy to implement in FORTH) to friends and colleagues. As added value my disk I/O can be faster.


\title{
Diegnoctice on Diak Buffore
}
```

SCR - 51
| SECTOR SKEW FCR CP/Fi FORHAT CLUSTERS
FORTH DEFIMITIOHS DECIMAL
: CTABLE ( bytesize TABLE )
<BUILDS O DS C, LONP NOES> + CB ;
22 16 104424 1% 12 6 26 20 14 R 2 21 15 9 3 23 17 11 5 25 19
1271027 CTARLE S-SKEW ( for CP/M clusters )
I:SETUP ( Setun n sectors for NXTS.)
adrs blk n .-- sec trk addr ... secn trkn addrn )
PNT IVER 12R * + ROT ROT OVER + I- SWAP I- SWAP
OS 1 26 /FIOS SKLP I+ S-SKEW SWAP ROT 12R - DUP
-1 +LMOP DRCIP :
: IRTS ( Read n sectors.) ( s t \#... sn tn an n mo- )
O CO SET-IO SEC-READ OISK-ERROR P IF LEAVE THEN LOOP:
6%
| B:CRE CP/"M Frimat nISK I/C
FORTH DEF MIITI!:SS OECI!:AL
\becauseTs ( virite n seetors tn C.P/K. elustar. )
.\mp@code{SET-!n SEC-!'R!TE DISK-ERRCP P IF LEAVE THE!' LMOP ;}
\because/\-CP!: 1 CD/!: stronen cluster I/0.1
aorirs bir f ... ) PR 52 + 2!?N /MMO SET-DRIVE
SEC/RLK USETUP ( 5% + Sn =lus*er alloc C!/a )
R) IF SEC./BLK N:RTS
ELSE SEC/RLY INTS
E:'?!F J!SK-ERRNR P \& ?ERPRR :
lll of scrcens 5! and a2 shanelessiv aden:ec from Jomn Jrons")
( fic-FiRT!! for the LSI-11.
15
ab

```


Timothy Hueng 9529 NE Gertz Circle Portlend, OR 97211

While I was In the procese of explaining the didking to some friends, I found it would be nice to show them some sort of representation which lists all the disk buffer status. This ahort program was then written for this purpose.

The figFORTH uses the memory above USER ares for the disk buffer. This disk buffer ares is further divided into several blocks with the length of each block equal to B/BUF +4 bytes. There are some implementations that set B/BUF to be 1024 bytes and some, like \(8080 \mathrm{CP} / \mathrm{M}\), that set it to be 128 bytes. Another constant beside B/BUF frequently referred in disking ts the B/SCR (buffers per screen). For B/BUF \(=1024\), the \(B / S C R=1\) and for \(B / B U F=128, B / S C R=8\).

Eech block needs 2 bytes in front of it at the header which contains the update bit (bit 25) end block number (lower 0-14 bitu). It also needs a 2 -byte tail to end the block.

The word BLOCK will put the beginning addrase of a given block (assuming the block number on stack before executing BLOCK ). With these simple words, virtual memory can be utilized, but it is beyond the scope of this short article.

The short program will display the status of each disk block until it is exheusted or you terminate it by pressing eny koy. The first thing it does is print out the title line (line 4). Line 5 sets up the boundary for the DO ... LOOP. Line 6 prints the buffer number while line 7 prints the beginning address of each buffer in hex. Lines 8 and 9 check the buffer update status. If it has been updated, then an ":" will be printed in the upd colurn. Lines 10 and 11 calculate the block number, screen number and the -sub number. The reason for teh -sub is because for my system, B/LBUF \(=128\). B/SCR \(=8\), there are 8 blocks to make a whole screen. So, I thought it would be hendier to know which subpart of a given screen the block I want.

Lines 12 and 13 check the early terminetion and finish the definition.

\title{
FLOATING POINT ON THE TRS-80
}

\author{
Kalman Fejes \\ Kalth Microsyatema \\ PO Box 5457, Station F \\ Ottawa, Ontario K2C 331 Canada
}

Most FORTH systems have no provisions for handling floating piont numbers, although most popular micros have the necessary routines hidden in their ROMbased BASIC interpreter. These are fast routines written in assembler. The following is to demonstrate how these can be accessed and used to implement single precision floating pint arithmetics for the TRS-80 in MMSFORTH, Version 1.8.

Single precision floating point data is stored as a normalized binary fraction, with an assumed decimal point before the most significant bit. The most significant bit also doubles as a sign bit.

A binary exponent takes one byte in each floating point number. It is kept in excess 128 form; that is, 128 is added to the actual binary exponent needed.

The binary mantissa is 24 bits long, the most significant bit representing the sign bit. It is stored as 3 bytes normally with the least significant byte (LSB) stored first and the most significant byte (MSB) last, followed by the exponent.

Numbers should be entered using the notation specified for the TRS-80 L2 BASIC. Integers and dobule precision numbers are converted to and stored internally as single prec ision numbers.

The complete vocabulary and listing of the source screens for either MMSFORTH or figFORTH (specify) is available for \(\$ 7\) (U.S.) from Kalth microsystems. It includes both single and double precision, trigonometric and log functions, floating point constant, variable and stack operators, conversion routines to/from integers (FORTH type) and floating piont numbers.

GLOSSARY
Single Precision Floating Point
\begin{tabular}{lll}
\(F+\) & \((F 1 F 1-F)\) & Add \\
& \((F=F 2+F 1)\) & \\
\(F-\) & \((F 2 F 1-F)\) & Subtract \\
& \((F=F 1-F 1)\) & \\
\(F *\) & \((F 2 F 1-F)\) & Multiply \\
& \((F=F 2 * F 1)\) & \\
& \((F 1 F 1-F)\) & Divide \\
& \((F=F 2 / F 1)\)
\end{tabular}\(\quad\)\begin{tabular}{l} 
F/
\end{tabular}

BLOCK 9


F \#
\(-F\)
Takes a number from the current buffer, converts it to single precision floating point number and leaves it on the stack.

F\#IN ( \(\quad\) - F)
Asks for a floating pint number from the keyboard, and leaves it on the stack.
\(F\) @ \(\quad(A-F)\)
Floating point fetch. Takes a floating point number from memory at address and leaves it on the stack.

F: (FA--)
Floating point store. Stores the floating point number on stack in memory at location \(A\).

\section*{F TEST ( - )}

A sample program to demonstrate the use of these floating point operators. It asks for a floating point number from the keyboard, manipulates it using all the operators defined and prints the result. (It should be the same number that was supplied.)

Notes: A -- 16 bit address
\(F, F 1, F 2\)-- are single precision floating pint numbers (two 16-bit words each).

\title{
TURNING THE STACK INTO LOCAL VARIABLES
}

\author{
Marc Perkel \\ Perkel Software Systems \\ 1636 N. Sherman \\ Springfield, MO 65803
}

Occasionally in writing a definition, I find that I need to do unwieldly stack juggling. For example, suppose you come into a word with the length, width, and height of a box and want to retum the volume, surface area, and length of edges. Try it!

For this kind of siuation 1 developed my ARGUMENTS-RESULTS words. The middle block fo the triad shows my solution to the box problem.

The phrase "3 ARGUMENTS" assigns the names of local variables 1 through 9 to nine stack positions, wtih S1, S2 and S3 returning the top 3 stack values that were there before 3 ARGUMENTS was executed. S4 through 59 are zero-filled and the stackpointer is set to just below 59 .

Sl thorugh S9 act as local variables returning their contents, not their addresses. To write to them you precede them with the word "TO ". For example, 5 TO 54 writes a 5 into \(\$ 4\). Execution of S4 returns a 5 to the stack.

After all calculating is done, the phrase "3 RESULTS" leaves that many results on the stack relative to the stack position when ARGUMENTS wes executed. All intermediate stack values are lost, which is good because you can leave the stack "dirty" and it doesn't matter.
```

SCR * B (***{ ARGUMENTS-RESULTS )***)
VARIARLE [ARG] UARIABLE [TO]
: +ARG CREATE , DOES` [ [ARG] [ SWAF - [TO] P ?GUF
IF O<IF +! ELSE ! ENLIF ELSE E ENIIF O [TO] ! ;

```

```

10 HARG 59 ( *TO VARIABLES* )
: TO 1 [TO] 1 ; ( WSETS STORE FLAG FOR +ARG* )
: +TO -1 [TO] ! ; ( *SETS +STORE FLAG FOK +ARG* )
: ARGUMENTS R` [ARG] O \R OR 2\# SFP + DUF [ARG] ! 12 - SFC SWAF
- 2/0 [1O O LOOF O [TO] ! ;
: RESULTS 2* [ARG] SWAF - SFE - 2/
O DO DROF LOOF R: R% [ARG] ! \R ;

```
SCR ©
    ( ARGUMENT EXAMFLE -- EOX COMES IN WITH HEIGHT, LENGTH
    8 WIDTH AND LEAVES VOLUME, SURFACE AREA \& LENGTH OF EDIGES)
    : HOX 3 AFGUMENTS
        (VOLM) S1 52 53**TO S4
        (SURF) S1 S2 2**S2 S3 2**S1532** + + TO 5!
        (EDGE) S1 \(4 * 524 * 534 *++\) TO 53
            55 TO S2
    S4 TO 51
            3 RESULTS ;


\title{
GRAPHIC GRAPHICS
}

California College of Arts and Crafts

Accompanying these comments are seversl graphic specimens drawn on Apple computer using FORTH and printed on a dot-matrix printer. They range from logotype design to experiments in geometry and pattern. One can generate real-time motion graphics on the Apple in which color and action partially compensate for the low resolution of 280 by 192 pixels. Hardcopy, whether prinout or color photo, isn't the final product. The interactive, sequenced and timed display on the screen is the designed product, likely to displace the medium of print on paper in the future.

While these graphic samples could have been programmed in other languages, 1 have found the advantages of using FORTH are both practical and expressive: immediate and modular experimentation with the peculiarities and limitations of the Apple video display, and orchestration of complex visual effects with self-named procedures rather than the tedious plots and pokes to undistinguished addresses. With this ease of wielding visual ideas, FORTH might lead to a new era of computer graphics, even creative expression.

It may remain individual and personal expression, however, without graphics standards. Transportability of grahics-generating code may be neither possible nor desirable considering the differences in video display generation, alternate character sets, shape tables, display lists, interrupts, available colors, etc., between microcomputers. Each has some individual features to exploit. Most have, however, such limited memory for graphics as to make machine-dependent economy an overriding aspect of programming for graphics.

Despite the rarity of FORTH graphics thus far, I 'm convinced it is an excellent vehicle for bringing out undiscovered graphics potential of each micro. In addition, the visibility gained by some effort to evolve grahic ideas in FORTH would heip in both spreading and teaching the language. Perhaps this issue of FORTH DIMENSIONS will stimulate just such activity.

Editor's Note: The author tells me that Osborne/McGraw-Hill publishers have used his patterns, generated on Apple Il using Cap'n Software FORTH, as cover artwork for their book "Some Common BASIC Programs':





\section*{CASES CONTINUED}

Editor's Note: In Volume Il, Number 3, FORTH DIMENSIONS published the results of FIGb CASE Statement Contest. As wo had hoped, the variety of responses hes stimulated further work on the subject. Here are four additional CASE constructes submitted by our readers.

\section*{Eskerts CASE for 8030}

John J. Cassady
Here is an 8080 (Z80) version of the keyed case statement by Charles Eaker that was published in FORTH DIMEN SIONS II/3, page 37. I have found it very use ful.
```

( CASE STATEMENT BY CHARLES EAKER FD II 3 39 JJC 81AUG09 )
: CASE ?COMP CSP ICSP 4; IMMEDIATE
CODE (OF) H POP D POP ' - 8 + CALL L A MOV H ORA O=
IF B INX B INX NEXT ENDIF D PUSH : BRANCH JMP C;
: OF 4 ?PAIRS COMPILE (OF) HERE 0 , 5 ; IMMEDIATE
ENDOF 5 ?PAIRS COMPILE BRANCH HERE O
ENDOF 5 ?PAIRS COMPILE BRANCH HERE O ,
ENDCASE 4 ?PAIRS COMPILE DROP
BEGIN SPE CSP E = 0=
WHILE 2 [COMPILE] THEN
REPEAT CSP ! ; IMMEDIATE
: TEST CASE 41 OF ." A " ENDOF
42 OF ." B " ENDOF
42 OF "" e "ENDOF ENDCASE;
( 41 TEST A OK )

```

\section*{Ekkert CASE Augmented}

Alfred J. Monroe 3769 Grandview Blvd. Los Angeles, CA 90066

1 was delighted with Dr. Eaker's CASE construction (FORTH DIMENSIONS, Vol. II, No. 3, p. 37) and implemented it immediately. Recently I have found it desirable to augment CASE with three sdditional constructs in order to treat ranges of variables. It has occurred to me thet other FORTH users may be interested in the same extension, hence this short note.

Screen 144 lists Dr. Eaker's CASE construct with one slight modification. OF has been modified to use (OF). The original OF compiled to ten bytes. The revised OF compiles to six bytes. This forty percent reduction in code \(b\) not as impressive as that which occurs using Dr. Eaker's CODE word (OF) construct, but it does have the advantage that it is highly portable. (OF) tests for equality and leaves a true or false flag on the stack. Note that it drops the test value if the test is true.

Screen 145 lists the extensions that I have found useful, <OF, >OF, and RNGOF. 〈OF does a "less than" test. >OF does a "greater than" test. RNG-OF does an inclusive range test. \(\angle O F\) and \(>O F\) are trivial modifications of OF and (OF). RANGE and RNG-OF are constructed in the same spirit as (OF) and OF.

Screen 144 compiles to 175 bytes. Screen 145 compiles to 223 bytes.
```

SCK 144

```

```

    : CHISE NCOM CSN !!SN 4 % IMMEDIHIL
    |LH, ULK = It UNCQ 1 ELSE G ENMIF's
    ```

```

            HEKE G, S 3 IMPHLDIHIE
    : ENHOU 5 FHHIKS CUNWILE ENHNCH HEKE G. SNWN
ICOMN1LES ENHIT 4; IMMEUIFIIE
: ENUCHSE 4 IMHIKS LINWILE UNXH EEGIN SNC CSN = = =
WHILE \& LCOMWILEJ ENOIF KEPERTT CSF: I IMMEDIFTE
Sck * 145

```



```

            HEKE G, % ; IMmalolHIE
    ```

```

- >UT 4 OHFINS CUHWILE (XOHS CONILE GUNHNCH
HEkE 6, % S IMMEDIFIE

```



```

    1M亚UIHIE
    -->
->

```
```

SCK 140
G EXHMHLLE USE OF FWOMMENTED CRSE )
4S LINSIFN:\ "G" 5T' CONSTFNT "g" 65 CONSTPWT "A"
| LONSIHNI "F" 1S CONSTFNT "CR"
- CUNSTANI CNTKL-C
G UHKIFHLLE FLFKS
: Sr'i-EkK LK ." SYHIMX ERKON, REENTER MMMMER " CK

```

```

    : "HBCHII CNIKL-C m IF DROF CR " COMNHNO ROCRT * CR CUIT
                        ELSE OLO ENOIF:
    -->
    14
    UE:-HEX LEMUL H HEX # UN TOF OF STFCK )
    H HFE UK. EFHKEK SOLIMIUN TO FAN INTENFR:TIUE TERNINFAL INFUII >
    ```

```

        1S = It 1 flafg! lN{UP
            ELSE LXP "G" < IF SYN-ERR
                ELSE DUF "Y" > IF DUP "A" < IF SYN-ERF
            ELSE LUHF "F"> IF SWN-ERKR ENGIF
            ENHIF ENHIT ENDIF ENDIF
    HLHIG fa= IF 4G - UUH y > IF % - ENOIF SNWP 16 * + ENDIF
        HLFGS OMNTIL:
    SN \# 14%
H fltFIEE: SCLLIIION IO THE TERMINFLL INWUT ROUTINE )
: 施;-HEN OHLHS:
2H EEGIN KEY WUH UUH EMIT

```

```

            CHEL LNIKL-G CO C-FRECRT, ENHOF
            CHEL LNIKL-G OF C-FEECRT, ENOCN
            CHEL LNIKL-G CF C-FEECRT, ENONOF
            CHEL LNIKL-G CF C-FEECRT, ENONOF
        ENLCASE
    H-His es G= it swffF 16:* + ENOIF
    LLH,J UNIIL;
    1%
    1 4
    =1K \$ 14F
\& SIILL NEFIEK B(\&UIIUN)
\#E:HEXX 的 HLHGS !
CHSE LHAIFL-C CF C-FHECKT ENOCF
"CK" OT 1 FLFHS! UKOF ENDOF
G:" "g" RNGGOF 48- ENOO
"A" "F" KHKG-CH 55- ENDO*
SNH-EHK
ENUCGEE
FLTIS G= IF SWFF 16 * + ENOIF
FLMG \& UNIILS
11
11
12
14
13
M, (GR" OF 1 FLFKS! UNOF ENHOF

```
    Screen 147 illustrates pre-Eaker
    colution to the deaign of en interactive
    terminel input that placea a hexadecimal
    number on the stack, and which providas
    for error detection end error recovery. It
    is, of course written in my usual sloppy,
    unennotated, sami-readeble fachion.

Screon 148 offers a neater solution in terms of <OF and >OF. It is definitely more readable. Screen 149 offers a still neater colution in terms of RNG-OF.

Screen 147 compiles to 160 bytes, screen 148 to 176 bytes, and screen 149 to 144 bytes. Need I say more?

\section*{SEND A CHECK TO FIG TODAYI MAKE THIS YOUR BEGINNINGI RENEW NOW}


RENEW TODAY!

\section*{CASE as a Defining Word}

Dan Lerner
After reading the CASE conteat articles and looking for a simple function, I am compelled to submit a simple CASE statement. These words are fast to compile and execute, compact, simple, generate minimum code, and very simple. There is no error checking since the form is so simple the most novice programmer can use it.

CASE is analogous to vectored GOTO in other languages. Its usage with my words is:
\begin{tabular}{lll} 
CASE & & NAME \\
A IS & FUNCTION A \\
B IS & FUNCTION B \\
C IS & FUNCTION C \\
(etc.) &
\end{tabular}

OTHERS ERROR FUNCTION
General usage would be as a menu selector; for example, you print a menu:
\begin{tabular}{ll}
1 & BREAKFAST \\
2 & LUNCH \\
3 & DINNER
\end{tabular}

SELECTION --)
The user types a number which goes \(n\) the stack, then executes the CASE word MEAL. MEAL selects BREAKFAST, LUNCH or DINNER, or ABORTS on error. The source is:
\begin{tabular}{cll} 
CASE & & MEAL \\
1 & IS & BREAKFAST \\
2 & IS & LUNCH \\
3 & IS & ONNER \\
OTHERS & NO MEAL
\end{tabular}

You have previously defined BREAKFAST, LUNCH, DINNER and NO MEAL.

MEAL

LUNCH DINNER NO MEAL


\section*{How CASE is Structured}

CASE builds an array using IS and OTHERS to fill and complete the values in the array. At execution, the DOES> portion of CASE takes a value from the stack and looks through the array for it. A match executes the word, no match executes the word after OTHERS in source.

The form of CASE is a new class of words, as CONSTANT, VARIABLE , MSG, etc. are. The code executed to test the array is minimal.

THIS IS THE END! THE END OF VOLUME III THE END OF YOUR MEMBERSHIP? DON'T LET IT HAPPENI RENEW TODAY

\section*{Genaralized CASE Stucture in FORTH}

\author{
E.H. Fey
}

\section*{Introduction}

The CASE CONTEST held by FIG last year ended with some excellent contributions to the FORTH literature. The judges noted however that few people tried to devise a general case structure encompassing both the positional type, where the case is selected by an integer denoting its position in the list of cases (ala FORTRAN's computed GO TO), and the more general keyed type of structure, where the case selector key is tested for a match in the case words key list.

This article discusses a general case structure which combines the positional and keyed types. Like FORTH itself, the case structure is extensible. I have added a third type called range where the case selector key is tested to be within the range of pairs of values in the case words key list.

For any of the three types of structures, the user is also provided with the option of using headerless high level code sequences to specify the execution behavior of the individual cases.

A complete source listing in figFORTH is given on screens 165 to 180 with illustrative examples on screens 180 and 181 . The source code listings may seem lengthier than usual but it is the author's practice to include the Glassary definition right with the source and to annotate the source code with notes on the status of the parameter stack. When this practice is followed, I find FORTH to be an emminent ly readable language, even months after the particular coding has been prepared. However, this style of coding requires a good FORTH video editor. With a good case structure in FOR TH, that is not difficult to develop.

\section*{Background}

In the Aug. '80 issue of Byte, Kim Harris introduced a very simple positional type of case compiler. A slightly revised version of his compiler is
: CASE: 〈LIST DOES〉 IX @ EXECUTE;

\section*{where}
: <LIST <BUILDS SMUDGE :CSP ] ; : IX (kpfa...adr) SWAP 1 MAX

1-DUP + + ;
and is used in the form:

CASE: \(x x x x\) cfal cfa2 .... ctan;
to define a case velector word named xXxX.

When the new word, \(x \times x\), in executed In the form
\[
k \times x \times x \quad(k=1,2, \ldots, n)
\]
the \(k^{\prime}\) th word in the list will be executed. For example, defline the following words, COW , CHICK , PIG , and BARN :
\[
\begin{aligned}
& \text { : COW } \text { " "MooOOOO" }^{\text {: CHICK }} \text { " Peep" ; } \\
& \text { : PIG "OInk"; } \\
& \text { CASE: BARN COW' PIG CHICK ; }
\end{aligned}
\]

If we now execute the sequence 2 BARN, Oink will be typed. Similarly 1 BARN will type MooOOoo.

Although there are no error checks, this case structure is easy to use, executes fast and requires a minimum of dictionary space for each case word, xxx. Bilobran, etal have used CASE: extensively in developing a FORTH file system with named record components (1980 FORML proc. pp 188, Nov. 1980). I have done likewise following their example.

The interesting part of the definition of CASE: is the <BUILDS part which I have called <LIST for obvious reasons. It creates the dictionary entry for \(\times x \times x\). Then, after executing SMUDGE and:CSP which are part of fig-FORTH's compiler security, it executes ] which forces FORTH into the compilation state so that the user cen enter the list. The list is terminated by \(;\) which completes the definition of \(x \times x \times\).

For CASE: words, the list is a list of code field addresses of previously defined FORTH words. Since FORTH is in the compilation state when the list is being entered, all the user has to do is list the names of the case select words ( COW PIG CHICK in the example of BARN). FORTH then compiles their code field addresses, as long as they are not special IMMEDIATE words which execute during compilation.

Now suppose that we knew beforehand that the code field address of PIG was say 14382. The same definition of BARN could then have been achieved by

CASE: BARN COW [14382, ] CHICK ;
where [ stopped the compilation state, 14382 was entered to the stack, the word, (comma), compiled it and 1 resumed the compilation state.

The point is that <LIST is a powerful word for entering named lists and dats of all sorts to the dictionary. The method of retrieval of the data is determined by the

DOES part of the compiler. Hence if we simply change the definition of the DOES part of CASE: , we can transform it into a general purpose case compiler.

\section*{The Multi-Pupone Cese Compiler}

The method utilized to develop a generalized case compiler is to compile a number for the case type as the first byte in the parameter field of \(x \times x\). At execution time, the number is retrieved and used to eelect the appropriate DOES> part for the case type of xxxx. The type number is transparent to the user.
```

The dofinition of the new case compiler is:
: MCASE: <BUILDS SMUDGE :CSP HERE $1 \mathrm{C}, 0 \mathrm{C}$, ]
DOES> DUP C@ DOESPART ;

```
where DOESPART is a case selector word defined by CASE: .

The <BUILDS part of MCASE: compiles a "l" for the default case type (positional) and a " 0 " for the count of the number of cases entered into the case list. It also leaves the parameter field address of the newly defined word on the stack so that it can be found later during the compilation process even though its name field is smudged.

If the newly defined case word, say \(x \times x x\), is to be other than the positional type, it is immediately followed by the word KEYED or RANGE to define the type of xxxx as keyed type \(=3\) or range type \(=5\).
\begin{tabular}{llll} 
: KEYED & 3 & OVER & \(C:\) IMMEDIATE \\
:RANGE & 5 OVER & \(C:\) IMMEDIATE
\end{tabular}

The case list subsequently entered must agree with the case type specified.

Two options are provided for the execution elements of the case list. The first or default option is the single word execution as in CASE: . The second option allows a headerless sequence of FORTH words to be defined as the execution elements of each case. The two may not be mixed.

A default case at the end of the case list is mandatory, although it may be a null word. The default case must be preceded by the word DEFAULT: whose definition is

\section*{: DEFAULT: ?COMP EOL, HERE OVER C@ [DEF] ; IMMEDIATE}
where EOL is an end of list terminator constant defined by
';S CFA CONSTANT EOL
and［DEF ］is a case selector word defined by CASE：．

DEF AULT：first checks to see that you are in the compile state since you should be compiling \(x x x x\) ．It then enters the end of list terminator，EOL ，to the diction－ ary．Finally it takes the parameter field address of xxxx left on the stack by the ＜BUILDS part of MCASE：，gets the type of \(\times \times \times x\) and executes the case relector word［DEF ］depending on the type of \(x \times x \times x\) ．If the type is 1,3 or 5 ，［DEF］counts the number of cases entered and stores it in the second byte of the parameter field of \(x \times x x\) ．If the case type is 2,4 or 6 ，then the execution elements are headerless code sequences．Hence for these types， ［DEF］initiates the process of defining the default code sequence．

\section*{Execution of Case Selector}

All case selector words，xxxx，defined by MCASE：are executed in the form：

\section*{k \(\mathbf{x x x x}\)}
where the key，\(k\) ，is an integer．The interpretation of \(k\) in selecting the case depends on the case list type．

With three case list types and two options for each type，there are actually 6 different forms of case lists availeble． Let＇s consider first the lists with aingle word execution elements．

\section*{Single Wand Execution Elemente}
（1）
Positional type
MCASE：is used in the form：
```

MCASE: xxxx cfal cfa2 ... cfan
DEFAULT: cfad ;

```

When xxxx is executed in the form \(k\) xxxx，the case cfak will be selected if \(k=1,2, \ldots, n\) ．Otherwise the default case，cfad，will be selected and executed．
（2）Keyed type

\begin{tabular}{cc}
165 & 0 \\
165 & 1 \\
165 & 2 \\
165 & 3 \\
165 & 4 \\
165 & 5 \\
165 & 6 \\
165 & 7 \\
165 & 8 \\
165 & 9 \\
165 & 16 \\
165 & 11 \\
165 & 12 \\
165 & 13 \\
165 & 14 \\
165 & 15 \\
166 & 0 \\
165 & 1 \\
165 & 2 \\
160 & 3 \\
166 & 4 \\
165 & 5 \\
166 & 6 \\
166 & 7 \\
166 & 8 \\
166 & 9 \\
166 & 10 \\
166 & 11 \\
166 & 12 \\
166 & 13 \\
165 & 14 \\
166 & 15 \\
167 & 0 \\
167 & 1 \\
167 & 2 \\
167 & 3 \\
167 & 4 \\
167 & 5 \\
167 & 6 \\
167 & 7 \\
167 & 8 \\
167 & 9 \\
167 & 10 \\
167 & 11 \\
167 & 12 \\
167 & 13 \\
167 & 14 \\
167 & 15 \\
169 & 0 \\
168 & 1 \\
168 & 2 \\
168 & 3 \\
168 & 4 \\
163 & 5 \\
168 & 6 \\
168 & 7 \\
168 & 8 \\
168 & 9 \\
168 & 10 \\
168 & 11 \\
168 & 12 \\
168 & 13 \\
168 & 14 \\
169 & 14 \\
169 & 3 \\
169 & 4 \\
169 & 5 \\
169 & 6 \\
169 & 7 \\
169 & 8 \\
169 & 9 \\
& \\
169
\end{tabular}

\section*{（ general case structure}

EHF 10／23／81
）
（ Execution variables ang akrays aje Kin Harris，Byte Aus ge ）
（ pf 184 siso see M．A．HCCourt，FD II／4 pp 109．EHF \(\mathrm{i} / 11 / 81\) ）
```

: Ix (kffa...ces:)(Compules adr of inde: k=1,2,···..fi)
SHAF 1 MAX ( ...ffa heaxi)
1- DUE + + ; (...ffat2[k-1])
: <LIST (Genergi <aulidS woro to construcl named lists)
ibuilaS SMuUGE !CSF ];
* CFA ( CONSTANT COLON (For heaberless coje definitions)
* iS CFA CONSTANT EO'. ( End of list delimiter )
: CASE: \&LIST DDES> IX Q EXECUTE ;
(Uses in the form: CASE: %%%% cfal cfaz...cfan; )
i to create ar, exechtion arras xxy% with inmtioi volues cfal:;
( cfo2,..fcfan which are cooe f2ejo acoresses of freviousls;)
( defaned words. Exeautins x:%x in the fo:m: k wox%
(will proouce the execution of cfak, }x=1,2,···,.,

```
    : LIST: <LIST DDES〉 IX e ;
    (Used in form: LIST; xxxx [ ni , ni, nJ ,...., ]; )
    ( to create list of constants named xxix . Executine xxa\% ;
    ( in the fore: \(k\) xxxx will leave rik on the stack.
: XEQUAS: 《LIST DOES〉 \(Q\) EXECUTE ;
    \(\{\) Used in the fors: XERUAK: x×xis cfa ;
    ( to create an execution variable xxxx with afi initioi vajue ;
    ( efa which is an existins word. Executins xixix ceuses ) -->
    ( cfa to be exesuleci. The wor of cfa bey be cherised by usiris,
    ( INSTALL nnnn AT xxxos where nnin is the new word. ;
: INSTALL ( . . cfa) [CDAFILE], STATE E IF COMPILE CFA ELSE CFA
    THEN ; IMMEDIATE
: at (cfa...) [COMPILE], State e if cohfile \(2+\) Carpile !
    ELSE 2t! THEN ; IHMEDIATE
: (ATKIN) (K Cfa pfa....) KOT 1 MAX \(2 \ldots+\) ! ( (Stores cía at )

: ATKIN ( \(k\) cfa....) [COHPILE]. STATE E IF COMPILE (ATKIN)
    ELSE (ATKIN) THEN; IMMEDIATE
    (Used in form: \(k\) IHSTALL cfa ATKIN 3xxx, )
    ( where xxxx is an execution array offined by CASE: , cfa) --,
    ( is the new word to be installed as eienent \(k=1,2, \ldots, n^{\prime}\) )
: Du4 : -->
( NOTE: NeCourt's imelementation of the functiga INSTALL ATKIN.
( does not wark inside : definition. The above does.
    hCASE: , A CEMERALIZED EXTENSIOM DF CASE:
    1. Three types of case stuchures:
            3. POSITIONAL ( derault)
            b. KEYED
            c. RaNGE
2. Two structure oplions for each lyfe:
            a. SINELE UDRN EXECUTION ( defabll)
            b. HIGH LEVEL HEADERLESS COLE SEQUENEE
    (Define boEspart and [DEf] as Execution arrays to be pilled)
    (in later)
    CASE: DOESFART DUM DUM DUM DUM DUM DUM DUN: 16 Cases
    CASE: [DEF] DUM DUM DUK DUM DUM DUM DUM;
    : NCASE: ( The seneralized case combiler )
    (BUILDS SHUDGE ICSP HERE (Leave pfa on stack)

\section*{(3) Range type}
```

MCASE: xxxx RANGE
[L1,H1,] cfal
["Ln, Hn,] cfan
DEFAULT: efad;

```

For this type each of the \(n\) entries to the case list consists of a pair of values specifying the upper and lower limits of the range, Li and Hl , followed by the execution element, cfai.

When xxxx is later executed in the form \(k \times x x x\), the case cfai will be selected if the condition
```

Li}<=k<= H

```
is found during a search of the list. If not, the default case, cfed, will be executed.

Hoaderlow Code Execution Elements
Instead of specifying the execution elements as previously de fined FORTH words, the elements may be specified as a sequence of FORTH words in the form:
\(H\) \(\qquad\)
or as
DEFAULT: .....seq.... ;
where ....seq.... is the sequence of executable FORTH words.

Again we have the three applicable case list types, the default type, position, the keyed type and the range type. Examples of the structure of each of these types is
(1) Positional type


（3）

\[
\begin{aligned}
& \text { MCASE: } x \times x \times \text { RANGE }
\end{aligned}
\]
\[
\begin{aligned}
& \text { ["Ln, H , ] H: ...seqn... } ; H \\
& \text { DEFAULT: ...seqd... ; }
\end{aligned}
\]

The interpretation of \(k\) in case selecting is the same as previously discussed for the single word execution of the same case type．The only difference is that a FORTH sequence，．．．seqi．．．is executed instead of a single FORTH word， cfai．

\section*{Examples}

Examples of all 6 possible combinations of case structures are given on Screens 180 and 181．If the screen is toaded and examples tested，typical execution results should be：
\begin{tabular}{|c|c|}
\hline EXECUTE & RESULT TYPED \\
\hline 1 BARN & MOO \\
\hline 2 BARN & OINK \\
\hline 18 BARN & PEEP（Default） \\
\hline 1200 & PEEP PEEP PEEP \\
\hline 3200 & PEEP PEEP MOO \\
\hline －6 200 & OINK OINK OINK \\
\hline \multicolumn{2}{|l|}{（Default）} \\
\hline 1 FARM & OINK（Default） \\
\hline 77 FARM & MOO \\
\hline －10 CASE & MOOOINK PEEP \\
\hline \multicolumn{2}{|l|}{（Default）} \\
\hline 77 CASE & MOO000000 \\
\hline －10 CORRAL & PEEP PEEP \\
\hline －1 CORRAL & OINK OINK \\
\hline 309 CORRAL & PEEP OINK MOO \\
\hline 310 CORRAL & MOO（Defeult） \\
\hline
\end{tabular}

\section*{COMMENTS}

1．Kim Harris＇case compiler，CASE： avoids the use of OVER \(=\mathbb{F}\) DROP ELSE．．．THEN for every case as used in many of the other CASE constructs． The result is shorter campiled code in the application．The compiler， MCASE：presented here is an extension of CASE：and consequently shares this feature．

2．The compiler，CASE：and the Execution Array introduced by M．A． McCourt in FD \(11 / 4\) pp 109 are functionally equivalent．Further，the Execution Variable，XEQVAR ，of McCourt turns out to be a degenerate case of CASE：with only one element in the case list．The definitions
：XEQARRAY CASE：；
：XEQVAR＜LIST DOES © EXECUTE ；

\section*{（ KEYEI TYPE WITK SINGLE WORD EXECUTION OPTION，TYPE 3 ）}
```

: KSIEF ( pfa adroef...) ( Counts cases entered and stores )
( in casecount at mfat1. AbJ'ess of defaull cfà is )
( adrcief=rfa+6+4[ri-1])
OVEF: 6 \& - 4/ it SWAE It C! ;
: KSCINLi ( K ffa...)( Saz:ches type 3 list for metch of Key )
( to K . Starts ai pfet2 . Exezutes cfa Efter metmhij ) --

```

```

        2f EEGIN 1 >R Wje e EJ_ - (...k ad!1 f)
            IF ( not EO') OUER OVER e = (...k as:1 k=?`)
                If (mbt=hej) 2t (...k ísil+2)
                ELSE F% 1 - 决 4 + THEN { ...N arembit )
            ELSE ( EOL ) 2t ( ...k eúroef
                THEN R; (...K arjrnew f)
    ```

    3 INSTA:L H.SFINL ATKIN DJESE'AST
    3 INSTALL KSLIEF ATKIN [EEFJ
( kEYEN GOTAO: WITH HIGH LEVEL IIGF If LIST, TYFE 4 )


        ( followins msith o: defeblit sesuence if no metatifojiú, )

                IF ( not EO. ) OVEF: DUEK \(\ell=(\ldots, k\) es' \(1 k=?\) ? )


                ELSE (EGL) 2t (...k Estojef)
                THEN K; ( ....K zo: ned f)
            UNTIL ( Matches a- EO-) SWAE bROF EXECUTE ;
    4 INSTALL NHEIND ATKIN DJESEATT
    4 INSTA゙心L FHEGF ATKIN [DEF] ( Sans as iSFE 2)


    (sto:e at ffati)
    OVER \(6+-6 /\) it SUAR it C! ;

    ( al 30tt2)

    : RSCINE ( \(K\) ffa....) ( Sestches tife 5 list for first oecurteri,
    ( ce of \(X\) withan faz of ranse vajues. Execules cfá foijun-


                If ( not EOL DVER DVEF KANET? (...K aj' 1 s;
                IF (in rance) 4 t ( ...k ár \(1+4\) )

            ELSE (EDI) 24 (, ...k arje Gef)
            THEN R) (, ..ok arjenew f)
            UNTIL ( In ranse or EOL) Sidar disos a ExEciJte ;
    S INSTALL RSEIND ATKIN DJESPART
    5 INSTALL RSDEF ATKIN [DEF]
（ Rance option mith high level dig in list，tyfe o ）
    : RHFIND ( \(K\) ffa...) ( Searches type 6 list for first oscurt-)
    ( ence of \(k\) within fair of ranse values. If found, execuies)
    ( followins hish level seavence, else exeautes dief seasence)

            IF ( nol ED'.) OVER OVER RANEE? (....K acirl s)
                IF (in ranse) \(6+\) ( .... k as \(1+6\) )
                ELSER: \(1->\mathrm{R} 4+\mathrm{E}\) THEN (....k agennt)
            ELSE (EAL) \(2 t\) ( \(\ldots k\) Edrgef)
\begin{tabular}{|c|c|}
\hline are flg-FORTH functional equivalonte & 17814 \\
\hline of McCourt's dafinitions. Hence & 17815 \\
\hline CASE: can be used as an Execution & 179 \\
\hline Array as suggested by McCourt. The & 179 \\
\hline definitions of AT, ATKIN and & 179 \\
\hline INSTALL on screens 167 and 168 cen & 179 \\
\hline be used ala McCourt to change the & 179 \\
\hline elements in CASE: list words. They & 179 \\
\hline are used in the form & 179 \\
\hline & 173 \\
\hline k INSTAL yyyy ATKIN xxxx & 179 \\
\hline & 179 \\
\hline to change the k'th element in a case & 17910 \\
\hline list, xxxx defined by C.ASE: to the code & 17911 \\
\hline field addrese of yyyy. Now whenever & 17512 \\
\hline \(k \times x \times x\) is encountered, the word yyyy & \(17 \%\) :3 \\
\hline will be executed rather than the & 17714 \\
\hline oriqinal word in the \(k\) 'th position of the & 179 : 5 \\
\hline case list. & 10's \\
\hline & 105 \\
\hline Using the previous CASE: exarnple of & 18, 2 \\
\hline HARN, if we execute & 10. \\
\hline & 102 \\
\hline 2 INSTALI. COW ATKIN BARN & 189 \\
\hline & 20\% \\
\hline the second case in BARN will be & 18: 7 \\
\hline changed from PlG to COW. Later & 18\% 8 \\
\hline execution of 2 BARN unywhere in the & 18; 9 \\
\hline prograin will then type MooOOoo & 18. 10 \\
\hline instesd of Oink. & 182 1: \\
\hline & 180 i2 \\
\hline Although this is non-structured & 10: 13 \\
\hline programming, it is still a valuable & 10214 \\
\hline programming tool when used & 12015 \\
\hline properly. The present definitions of & 1810 \\
\hline INSTALI. and ATKIN can be used & 18: 1 \\
\hline within a color definition. & 18i i \\
\hline & 1813 \\
\hline Please note that the use of the & 181 4 \\
\hline Execution Array in the development of & 181 \\
\hline MCASI: : on screen 169 is purely & \(18 i\) \\
\hline stylistic. It is not a necessary feature & 1817 \\
\hline of the development. & 1818 \\
\hline & 161 9 \\
\hline The essentially unique feature of & 16116 \\
\hline FORTH is that it is extendable by the & 16i 12 \\
\hline user. With an expanding F()XTTH & 10: 12 \\
\hline fiterature, it is clear to this euthor & 181 ; 3 \\
\hline that FORTH will improve with time & 18114 \\
\hline faster than all other lanquages and & 18115 \\
\hline
\end{tabular}
that there is no upper lirnit to its improveinent. It has heen less than 18 months since I first got + ORTH up and rurning. In that short period of time, thanks to the fiq literature, the FORTH system I have ruruing now is, in my opinion, vastly superior to any other lanquage I have ever seen. And it will qet hetter!
```THEN R
```



``` UTIL ( In rence o: EOL, SUAP DRJ. EXECUTE: \(-\infty\)
6 INSTALL RHESND ATKIN DIESPAST
C INSPALL PNBSF ATKIM [UEFJ ( Same os types 2 and 4)
18
( KNASE: EXATRLES )
: PIC "" OINN" ;
:COM "" MJJ";
: CMICK .* PEEP * ;
MEASE: BARN COU PIG CHICK DEFAJIT: CHICN:
MEASE: 20J H: CHICK CHICK CHICN IH
    H: coke" "\mp@code{ajJJ" iH}
    H: CHICN CHICN COW iH
    GEFAJLT: FIG PIC FIG:
NEASE: FAKM KEYED [ 82, ] PIG
                        [77,] COU
                        [ 67, ] CHICN -->
                        DEFAUST: FIG;
MEASE: CASE RIEYED [ 77 , ] H: COW "* OJJJJJ" iH
        [ 8), ] H: FIG FIG iH
        [ 6%,] H: CHICN CHICN ;H
        DNFA'ST: CO& FIG CiNICK ;
MSASE: PEN RANEE [ -32763 , -1, ] CHICl.
        [ [ 0, 1
        DEFAJET: ;
MCASE: CDRRAL RAKEE [ - 10, -5 , ] H: CHICK CHICN iH
        [ -1, -1, ] H: FIG SFANE FJG iH
        [ O. O. J H: COA SSANE COW IH
        〔 1. 309 , ] H: CHICN FIG CON iH DSFASLT: CON;

A FORTH Standards Team meeting will be held in Bethesda, MD, from May 11 through May 14. The meeting is open to the current Standards Team members and a limited number of observers. The site will be the National 4 H Center, a selfcontained educational facility, just outside Washington, DC. The campus-like Center has meeting rooms, dining facilities and cormitory accommodations.

This four-day meeting will allow world-wide Team members to consider proposals and corrections for the current FORTH Standard and develop future standards policy. Participation is possible by submittal and attendance. Written submittals received by April 30 will be distributed to attendees before the meeting. Late receipts will be distribued at the team meeting. Those wishing to attend must apply without delay, as space is severely limited.

Applicants (ather than team members) must submit a biography by April 15 for consideration by the credentials committee. You should include:
1. Your skills and comprehension of multiple FORTH dialects and the ir application.
2. Why your views are representative of a significant portion of the FORTH community.

Accommodations are \(\$ 41\) to \(\$ 47\) per day, per person, including meals. Send a refundable \(\$ 100\) deposit (and biography for observers) to the meeting coordinator. You will receive further details on choices in housing and meals.

Submittals are essential if Team actions are to represent the broadening scope of FORTH users. Specific consideration will be given to an addendum correcting FORTH-79, the Team Charter, and alliance with other standards groups. Those not attending may receive copies of submittals by sending \(\$ 30\) to the meeting coordinator.

All submittals and reservations should be directed to the meeting coordinator:

Pam Totta
Creative Solutions
4801 Randolph Road
Rockville, MD 20852
(301) 984-0262


\section*{LECTURES ON APPLIED FORTH}

\section*{- two day cominar on Forth and its application}
and the
1982 ROCHESTER FORTH CONFERENCE ON DATA BASES AND PROCESS CONTROL
May 17 through May 21, 1982
University of Rochester Rochester, New York

As part of the 1982 Rochester FORTH Conference on Data Basea and Process Control there will be a two day seminar on Appiled YorTh. Managere and programmers will find these lectures very uaeful for exploring Forth applications and programing concepta. Bach lecturer will also lead a Working Group at the subsequent Conference. Participants should have a copy of Leo Brodic's book, Starting FORTE, which is available from Mountain View Press, PO Box 4656, Mt. View, CA 94040 for \(\$ 16.00\).

Lecturers for the two day seminar are:
Leo Brodie, author of Starting FORTH, on "Beginning FORTH".
Xim Harris, of Laxen \& Harris, Inc., on "PORTB Programalig Style".
Hans Nieuwenhuijzen, of the University of Utrecht, on "PORTH Programing Environment".

Lerry Porsley, of the Laboratory for Leser Energetics, on "Extensible Control and Data Structure".
David Beers of Aregon Systems, Inc., on "A Large Programing Project Case Study: Building a Relational Database in Fortr".
Steven Marcus of Kitt Peak National Observatory, on "Assemblers \& Cross Assemblers".

James Harwood of the Institute for Astronomy at the University of Ravail, on "Computation Tradeoffs".
Roger Stapleton of St. Andrews Observatory, Scotland, on "Hardwere Control with FORTH".

Raymond Dessey of Virginia Polytechic Institute, on "Concurrency, Networking and Instrument Control".

REGISTRATION FORM
(must be received by April 23, 1982)
Name
Address
City \(\qquad\) State ZIP

Phone (Days) \(\qquad\) )
choices to be made
__ Applied FORTH Seminar, May \(17 \& 18\)
\(\$ 200.00\)
_ 1982 Rochester FORTH Conference, May 19-21
100.00
\(\qquad\) Housing for: (circle dates) May 161718192021
\$ 13.00/person dbl 16.50/person agl

TOTAL AMOURT ENCLOSED \$ \(\qquad\)
Make checks payable to: "University of Rochester/FORTE Conference"
Send check and Registration to:
Mrs. B. Rueckert, Lab for Laser Energetics, 250 E River Rd, Rochester, WI 14623
For information call: Barbara Rueckert (716) 275-2357

\section*{NEW PRODUCTS}

\section*{Marx FORTH for Northstar} now Available

Marx FORTH is a fast, powerful FORTH system written in Z-80 code. Package includes self-compiler, complete source code, screen editor, and "smart" assembler. Some of the features include calls to the \(N^{*}\) directory functions allowing creation, deletion and listing of directories and ease of writing FORTH programs that operate on files created bs N* BASIC. Some of the performance features include very fast compile speeds, very fast math, 31 -character variable length names, case compiler security, arguments-results, link field in front of name, and many machine code definitions for high speed.

The self-compiler allows you to change anything. If you don't like how I do it, change it! Add anything you want. Price is \$85 on \(N^{*}\) single density diskette. Source listing available separately for \(\$ 25\).

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

\section*{FORTH Programming Aide}

FORTH Programming Aids are high level FOR TH routines which enhance the development and debugging of FORTH programs and complement cross compiler and meta compiler operations with the following features:
- A command to decompile high level FORTH words from RAM into structured FORTH source code including structure control words. This command is useful to examine the actual source code of a FORTH word, or to obtAln variations of FORTH words by decompiling to disk, editing, and recompiling the modified source code.
- A command to find words called by a specified word to all nesting levels.
- Commands to patch improvements into compiled words and to merge infrequently called words for increased program speed.
- Complete source code and 40-page manual are provided.

Requires a FORTH nucleus using the fig-FORTH model; a minimum of 3 K by tes and a recommended 13 K bytes of free dictionary space. \(\$ 150\) single CPU license; \(\$ 25\) for manual alone (credit applied toward program purchase). Califomia residents add 6.5\% tax. Add \(\$ 15\) for foreign air shipments. Available on 8 -inch ss/sd disks (FORTH screens or CP/M 2.2
file of screens), and Apple 3.2 and 3.3 disks; inquite about other formats.

\author{
Ben Curry \\ Curry Associates \\ PO Box 11324 \\ Palo Alto, CA 94306
}

\section*{Naw Book: Introduction to FORTH}

Introduction to FORTH, 142-page textbook by Ken Knecht, presents the most complete information available on the MMS FORTH version of the FORTH language. It is written for anyone who wants to leam how to write computer software using FORTH.

No previous knowledge of FORTH is required, but some exposure to Microsoft Level II BASIC will be helpful. Although the book is designed specifically for the MMSFORTH version of FORTH for the Radio Shack TRS-80 Models I and III, most program examples can be adapted to run on other microcomputers that use different versions of FORTH.

\section*{RENEW NOW}

\section*{FORTH for Ohio Selmitific}

We've received from Technical Producte Co. a copy of their newsletter. This issue contains product news and update screens for FORTH-79. We applaud thair intent of good customer support, but note technical errors in definition of soveral stendard words (WORD, R@ , END-CODE, 2CONSTANT , \(D\) ). This OS1-FORTH operates with Ohio Scientific OS_65D 3.3 operating system release.

Their now address is Technical Products CO., Box 2358, Boone, NC 28607--ed.

Henry Svec
668 Sherene Terrace
London Ontario Canada
NGH 3 Kl

\section*{MCZ, ZOS, UDS FORTH}

FORTH is now ruming on Zilog MCZ, ZDS, and Multitech UDS microcomputer systems. It has compiler, editor, aseembler, text interpreter, and I/O drives for floppy disk, Centronics printer, and RS232 devices.

Assembly source listing is available now for \(\$ 10\). Source code on diskette is \(\$ 50\) (specify MCZ, ZDS, or UDS). User's manual will accompany each order.

Send checks to Thomas Y. Lo, Electrical Engineering Department, Chung Yuan Christian University, Chung Li, Taiwan, Republic of China.

\section*{Software for OSI CIP}

Shoot The Teacher - Find the teacher and shoot him with your water pistol. (Teaches basic graphing) \(\$ 6.95\)

Speedo Math - Race the computer with your car. (Drjlls besic addition and multiplication) \$6.95

Kamakaze Education Pack - Four programs in one. Addition, \(X\) Tables, Spelling, and Place Value Drill. Answer a question and your men go on their last mission. \$11.95.

That's Crazy - A takeoff from a famous TV Show where you risk your life to jump over cars and a canyon, A spelling program that provides hours of entertainment. I \(\$ 11.95\) (specify grade level)

Went Ads Life Skills - A program that hetps slow readers understand the Want Ads. Five levels of difficulty. \(\$ 7.95\)

Rescue Ship - Transport injured soldiers to the hoapital. But the enemy has covered the ocean with mines. One of them could destroy you.

Addition - \(\$ 11.95\)
Subtraction - \(\$ 11.95\)
Multiplicstion - \$11.95
(all three on tape - \(\$ 28.00\) )
Please include \(\$ 1.00\) to cover postage and handling and send to:

\section*{FORTH VENDORS}

The following vendere heve verition of FORTH avalldble or ere contultante. (FIG makes no judgment on any producti.)
```

ALPHAL MORO
Professional Manegement Services
724 Arastradero Rd. ©109
Palo Alto, CA 94306
(408) 252-2218
Sierra Computer Co.
6 1 7 Mark NE
Albuquerque, NM 17123

```
APPLE
    IDPC Compeny
    P. O. Box 11594
    Philadelphie, PA 19116
    (215) 676-3235
    MS (Cep'n Software)
    281 Arlington Averwe
    Berkeley, CA 9470A
    (415) 525-9452
    Ceorge Lyone
    280 Henderton St
    Jerwey Clty, NJ 07302
    (201) 451-2905
    MicroMotion
    12077 Wilshire Blyd. 1506
    Loa Angeles, CA 90025
    (213) 821-4340
CROSS COMPM_ERS
    Nautilus Systerm
    P.O. Box 1098
    Sante Cruz, CA 95061
    (408) 475-7461
polyFORTH
    FORTH, Inc.
    2309 Pecific Conat Hwy.
    Hermose Beach, CA 90254
    (213) 372-8493
    LYNX
    3301 Oceen Park 301
    Senta Monica, CA 90405
    (213) 450-2466
    M \& B Design
    820 Sweetbay Drive
    Sumyvale, CA 94006
Micrupolis
    Shaw Lebs, Ltd.
    P. O. Box 3471
    Hayward, CA 94540
    (415) 276-6050
North Ster
    The Software Worke, Inc.
    P. O. Box 4386
    Mountain View, CA 94040
    (408) 736-4938
POP-11
    Laboratory Software Systerms, Inc.
    3634 Mandeville Canyon Rd.
    Loe Angelee, CA 90049
    (213) 472-6995
oss
    Consumer Computers
    8907 LaMesa BIvd
    LaMesa, CA 92041
    (714) 698-8088

Software Federation
44 Univeraity Dr.
Afllington Halghts, il 60034
(312) 259-1355

Technical Producta Co.
P. O. Box 12903

Geinaville, FL 32604
(904) 372-8439

Tom Zimmar
292 Falcato Dr.
Milipltes, CA 95035

\section*{1802}

FSS
P. O. Box 8403

Austin, TX 78712
(512) 477-2207

6000 \& 6009
Talbot Mierooyatorne
1927 Curtis Avenue
Redondo Eanch, CA 90278
(213) 376-9941

TRS-20
The Micro Worky (Color Computer)
P. O. Box 1110

Del Mar, CA 92014
(714) 942-2400

Miller Microcomputer Sarvicen
61 Leke Shore Rd.
Natick, MA 01760
(617) 653-6136

The Software Farm
P. O. Box 2304

Reaton, VA 22090
Sirius Systems
7528 Oak Ridge Hwy.
Knoxville, TN 37921
(615) 693-6583

\section*{6502}

Eric C. Rehrike
340 S. Rench View Clicle \(\$ 61\)
Anahoim Hille, CA 92087
Setum Software, Ltd.
P. O. Box 397

Now Westminister, BC
VYL \(4 Y 7\) CANADA

\section*{8000/Z80/CPM}

Leboratory Micronystems
4147 Beethoven St.
Los Angelen, CA 90066
(213) 390-9292

Timin Engineering Co.
9575 Genese Ave. FE-2
Sen Diego, CA 92121
(714) 455-9000

Application Packeges
Inno5ys
2150 Shattuck Avenve
Berkeley, CA 94704
(415) 843-8114

Decision Resourcen Corp.
28203 Ridgefern CL.
Rancho Palo Verde, CA 90274
(213) 377-3533

60000
Emperical Res. Orp.
P. O. Box 1176

Millton, WA 98354
(206) 631-4855

Firmware, Boarto and Machinee
Datricon
7911 NE 33rd Dr.
Portiand, OR 97211
(503) 284-8277

Forward Technology
2595 Martin Avenue
Sente Clara, CA 95050
(408) 293-6993

Rockwell International
Microelectronice Devices
P.O. Box 3669

Ancheim, CA 92803
(714) 632-2862

Zendex Corp.
6398 Dougherty Rd.
Dublin, CA 94S66
Vartety of FORTH Prodrets
Interective Computer Systems, Inc.
s403 Di Marco Rd
Tampa, FL 33614
Mountaln Viow Prese
P. O. Box 4656

Mountain Vlaw, CA 94040
(415) 961-4103

Suparsoft Amociates
P.O. Box 1628

Champaign, IL 61820
(217) 359-2112

\section*{Ceneuitente}

Creative Solutione, Inc.
4801 Rendolph Rd.
Rockville, MD 20852
Dave Boulton
581 Oakridge Or.
Redwood City, CA 94062
(415) 368-3257

Leo Brodie
9720 Beden Averus
Chateworth, CA 91311
(213) 998-8302

Go FORTH
504 Lakernesd Way
Redwood City, CA 94062
(415) 366-6124

Inner Accesm
517K Marine Vlew
Belmont, CA 94002
(415) 591-8295

Laxen \& Harria, Inc.
24301 Southiand Drive, 303
Hayward, CA 94545
(415) 887-2894

Microsystems, Inc.
2500 E. Foothill Blvd., 102
Passdena, CA 91107
(213) 577-1471

\section*{FIG CHAPTERS}

How to form a FIG Chapter:
1. You decide on a time and place for the first meeting in your area. (Allow at least 8 weeks for steps 2 and 3.)
2. Send FIG a meeting announcement on one side of \(8-1 / 2 \times 11\) paper (one copy is enough). Also send list of ZIP numbers that you went mailed to (use first three digits if it works for you).
3. FIG will print, address and mail to members with the ZIP's you want from San Carlos, CA.
4. When you've had your first meeting with 5 or more attendees then FIG will provide you with names in your area. You have to tell us when you have 5 or more.

Northern California
4th Sat FIG Monthly Meeting, 1:00 p.m., at Southland Shopping Ctr., Hayward, CA. FORML Workshop at 10:00 am.

Southem California
Los Angeles
\(\begin{array}{ll}\text { 4th Sat } & \begin{array}{ll}\text { FIG Meeting, 11:00 a.me Allstate } \\ \text { Savings, } 8800 & 50 . \text { Sepulveda, L.A. }\end{array}\end{array}\) Philip Wasson, (213) 649-1428.

Orange County
\(\begin{array}{ll}3 \text { rd Sat } & \text { FlG Meeting, 12:00 noon, Fullerton } \\ & \text { Savings, } 18020 \text { Brockhorst, Fountain }\end{array}\) Valley, CA. (714) 896-2016.

Sen Diego
Thur FIG Meeting, 12:00 noon. Guy Kelly, (714) 268-3100, \(\times 4784\) for site.

Northwest
Seattle Chuck Pliske or Dwight Vandenburg, (206) 542-7611.

\section*{New England}

\section*{Boston}
lst Wed FIG Meeting, 7:00 p.m., Mitre Corp., Cafeteria, Bedford, MA. Bob Demrow, (617) 389-6400, x198.

Boston
3rd Wed MMSFOR TH Users Group, 7:00 p.m., Cochituate, MA. Dick Miller, (617) 653-6136 for site.

Southwest
Phoenix Peter Bates at (602) 996-8398.
Tulse
3rd Tues FIG Meeting, 7:30 p.m., The Computer Store, 4343 So. Peoria, Tulsa, OK. Bob Giles, (918) 5999304 or Art Gorski, (918) 743-0113.

Austin John Hastings, (512) 327-5864.
Dallas
Ft. Worth 4th Thur

FIG Meeting, 7:00 p.m., Software

Automation, 1005 Business Parkway, Richardson, TX. Marvin Elder, (214) 231-9142 or Bill Drissel (214) 264-9680.

\section*{Mountain West}

Salt Lake City
Bill Haygood, (801) 942-8000
Mid Atlantic
Potomac Joel Shprentz, (703) 437-9218.
New Jersey George Lyons (201) 451-2905.
Now York Tom Jung, (212) 746-4062.
\(\frac{\text { Midwest }}{\text { Detroit }}\) Dean Vieau, (313) 493-5105.
Minnesota
Ist Mon FIG Meeting. Mark Abbott (days), (612) 854-8776 or Fred Olson, (612) 588-9532. Call for meeting place or write to: MNFIG, 1156 Lincoln Avenue, St. Paul, MN 55105.

Foreion
Australis Lance Collins (03) 292600.
England FORTH Interest Group, c/o 38, Worsley Road, Frimley, Camberley, Surrey, GU16 5AU, England

Japan
FORTH Interest Group, Bebabbldg. BF, 3-23-8, Nishi-Shimbashi, Minato-ku, Toyko, 105 Japen.

Canada - Quebec
Gilles Paillard, (418) 871-1960 or 643-2561.
W. Germeny Wolf Gervert, Roter Hahn 29, D-2 Hamburg 72, West Cermany,(040) 644-3985.```


[^0]:    Ralman Fejes
    KALTH MICROSYSTEMS
    P.O. Box 5457, Station $F$

    Ottawa, Ontario K2C 3JI
    Canada

[^1]:    Wife/Husband/Friend (\$85.00 for room and meals)
    I will arrive the afternoon or night before, please reserve a room for: \#_on Tuesday, November 24 @ _ $\$ 35.00$ double
    or $\qquad$ $\$ 47.00$ single

    FOR MORE INFORMATION CALL: ROY MARTENS (415) 962-8653

